

Lecture Notes on Modern Multivariate Statistical Learning-Version IV

Stephen B. Vardeman
Analytics Iowa LLC
and
Iowa State University

June 18, 2021

Abstract

This set of notes is the most recent reorganization and update-in-progress of Modern Multivariate Statistical Learning course material developed 2009-2020 over 7 offerings of PhD-level courses and 4 offerings of an MS-level course in the Iowa State University Statistics Department, a short course given in the Statistics Group at Los Alamos National Lab, and two offered through Statistical Horizons LLC. Early versions of the courses were based mostly on the topics and organization of *The Elements of Statistical Learning* by Hastie, Tibshirani, and Friedman, though very substantial parts benefited from Izenman's *Modern Multivariate Statistical Techniques*, and from *Principles and Theory for Data Mining and Machine Learning* by Clarke, Fokoué, and Zhang.

The present version benefits from a thoughtful set of written comments on an earlier iteration of the notes provided by Ken Ryan and Mark Culp, incisive observations on the material and suggestions concerning what I've said about it made by Max Morris and Huaiqing Wu during the MS-level course we taught together Spring 2014, additional helpful critiques offered by LANL statisticians in Summer 2016, and material from Bishop's *Pattern Recognition and Machine Learning*, *Applied Predictive Modeling* by Kuhn and Johnson, and *An Introduction to Statistical Learning* by James, Witten, Hastie, and Tibshirani. The work of a number of ISU PhD and MS advisees including Jing Li, Wen Zhou, Cory Lanker, Andee Kaplan, and Abhishek Chakraborty has also provided useful additional content reflected in this version.

These notes have as prerequisites the Statistical Theory, Methods, and Computing content of the first year courses in a Statistics MS program, though presumably much of them can be understood with less background.

Contents

I Introduction, Generalities, and Some Background Material	8
1 Overview/Context	8
1.1 Notation and Terminology	8
1.2 What is New Here (Particularly in Prediction)?	9
1.2.1 Matching Complexity to Training Set Information Content	9
1.2.2 The "Curse of Dimensionality"	10
1.3 Some Initial Generalities About Prediction	12
1.3.1 Representing What is Known: Creating a Training Set for Prediction	12
1.3.2 Theoretically Optimal (Unrealizable) Predictors	13
1.3.3 Nearest Neighbor Rules	15
1.3.4 General Decomposition of the Expected Prediction Loss for \hat{f}	16
1.3.5 A More Detailed Decomposition for Err in SEL Prediction and Variance-Bias Trade-off	18
1.3.6 Approximating Err and Cross-Validation	20
1.3.7 Choosing a Predictor Based on Cross-Validation	23
1.3.8 Penalized Training Error Fitting and Choosing Complexity	24
1.4 Good Features and Prediction	25
1.4.1 Classification Models and Optimal Features	25
1.4.2 Approximating "Partially Optimal" Numerical Features for Discrete Parts of Input Vectors	26
1.4.3 Abstract Feature Spaces (of Functions) and "Kernels"	28
1.4.4 Document Features and String Kernels for Text Processing	33
1.4.5 "Feature Engineering" and Data "Pre-processing": More Perspective and Prediction of Predictor Efficacy	36
1.5 Some More Generalities for 2-Class Classification	38
1.5.1 More on the Form of an Optimal 0-1 Loss Classifier for $K = 2$	38
1.5.2 Other Prediction Problems in 2-Class Classification Models	40
1.5.3 "Voting Functions," Losses for Them, and Expected 0-1 Loss	42
1.6 Density Estimation and Approximately Optimal and Naive Bayes Classification	44
1.7 Plotting to Portray the Effects of Particular Inputs in Prediction	49
2 Some Linear Theory, Linear Algebra, and Principal Components	50
2.1 Inner Product Spaces	50
2.2 The (General) Gram-Schmidt Process and the QR Decomposition of a $rank = p$ Matrix \mathbf{X}	52

2.3	The Singular Value Decomposition of \mathbf{X}	56
2.3.1	The Singular Value Decomposition and General Inner Product Spaces	57
2.4	Matrices of Centered Columns and Principal Components	59
2.4.1	"Ordinary" Principal Components	59
2.4.2	"Kernel" Principal Components	63
2.4.3	Graphical (Spectral) Features	64
 II Supervised Learning I: Basic Prediction Methodology		66
3	(Non-OLS) SEL Linear Predictors	66
3.1	Ridge Regression, the Lasso, and Some Other Shrinking Methods	67
3.1.1	Ridge Regression	67
3.1.2	The Lasso, Etc.	71
3.1.3	Least Angle Regression (LAR)	76
3.2	Two Methods With Derived Input Variables	79
3.2.1	Principal Components Regression	79
3.2.2	Partial Least Squares Regression	80
4	Linear SEL Prediction Using Basis Functions	83
4.1	$p = 1$ Wavelet Bases	84
4.2	$p = 1$ Piecewise Polynomials and Regression Splines	87
4.3	Basis Functions and p -Dimensional Inputs	89
4.3.1	Multi-Dimensional Regression Splines (Tensor Product Bases)	89
4.3.2	MARS (Multivariate Adaptive Regression Splines)	90
5	Smoothing Splines and SEL Prediction	92
5.1	$p = 1$ Smoothing Splines	92
5.2	Multi-Dimensional Smoothing Splines	97
5.3	An Abstraction of the Smoothing Spline Material and Penalized Fitting in \mathbb{R}^N	99
5.4	Graph-Based Penalized Fitting/Smoothing (and Semi-Supervised Learning)	100
6	Kernel and Local Regression Smoothing Methods and SEL Prediction	102
6.1	One-dimensional Kernel and Local Regression Smoothers	102
6.2	Local Regression Smoothing in p Dimensions	105
7	High-Dimensional Use of Low-Dimensional Smoothers and SEL Prediction	106
7.1	Structured Regression Functions	106
7.1.1	Additive Models	106
7.1.2	Other Structured Regression Forms	107

7.2	Projection Pursuit Regression	108
8	Highly Flexible Non-Linear Parametric Prediction Methods	108
8.1	Neural Network Regression	108
8.2	Neural Network Classification	110
8.3	Fitting Neural Networks	111
8.3.1	The Back-Propagation Algorithm	111
8.3.2	Formal Regularization of Fitting	114
8.4	Convolutional Neural Networks	115
8.5	Recurrent Neural Networks	118
8.6	Radial Basis Function Networks	119
9	Prediction Methods Based on Rectangles: Trees and PRIM	120
9.1	Regression and Classification Trees (CART)	121
9.1.1	Regression Trees	121
9.1.2	Classification Trees	123
9.1.3	Optimal Subtrees	124
9.1.4	Measuring the Importance of Inputs for Tree Predictors	127
9.2	PRIM (Patient Rule Induction Method)	127
10	Predictors Built on Bootstrap Samples	129
10.1	Bagging in General	129
10.2	Random Forests: Special Bagging of Tree Predictors	131
10.3	Measuring the Importance of Inputs for Bagged Predictors	133
10.3.1	The Boruta Wrapper/Heuristic for Variable Selection	134
10.4	Bumping and "Active Set Selection"	135
11	"Ensembles" of Predictors	136
11.1	Bayesian Model Averaging for Prediction	136
11.2	Stacking: SEL ... and 0-1 Loss	138
11.3	"Generalized Stacking" and "Deep" Structures for Prediction	140
11.4	Boosting/Successive Approximation	143
11.4.1	SEL Boosting	143
11.4.2	General "Gradient Boosting"	144
11.4.3	Some Issues Related to Boosting Practice	147
11.4.4	AdaBoost.M1	148
11.5	Quinlan's Cubist and "Divide and Conquer" Strategies	152
III	Intermission: Perspective and Prediction in Practice	154
IV	Supervised Learning II: More on Classification and Additional Theory	156

12 Basic Linear (and a Bit on Quadratic) Methods of Classification	156
12.1 Linear (and a bit on Quadratic) Discriminant Analysis	157
12.1.1 Dimension Reduction in LDA	159
12.2 Logistic Regression	161
12.3 Separating Hyperplanes	165
13 Support Vector Machines	166
13.1 The Linearly Separable Case: Maximum Margin Classifiers	166
13.2 The Linearly Non-separable Case: Support Vector Classifiers	170
13.3 SV Classifiers and Kernels: Support Vector Machines	173
13.3.1 Heuristics	173
13.3.2 A Penalized-Fitting Function-Space Optimization Argument	175
13.3.3 A Function-Space-Support-Vector-Classifier Geometry Argument	177
13.3.4 Some Perspective on SVMs	178
13.4 Other Support Vector Methods	179
14 Prototype and (More on) Nearest Neighbor Methods of Classification	181
15 Reproducing Kernel Hilbert Spaces: Penalized/Regularized and Bayes Prediction	184
15.1 RKHSs and $p = 1$ Cubic Smoothing Splines	184
15.2 What is Possible Beginning from Linear Functionals and Linear Differential Operators for $p = 1$	185
15.3 What Is Common Beginning Directly From a Kernel	187
15.3.1 Reprise of Some Special Cases	192
15.3.2 Addendum Regarding the Structures of the Spaces Related to a Kernel	193
15.4 Gaussian Process "Priors," Bayes Predictors, and RKHSs	194
16 More on Understanding and Predicting Predictor Performance	196
16.1 Optimism of the Training Error	197
16.2 C_p , AIC and BIC	198
16.2.1 C_p and AIC	198
16.2.2 BIC	198
16.3 Cross-Validation Estimation of Err	200
16.4 Bootstrap Estimation of Err	201
V Unsupervised Learning Methods	201

17	Some Methods of Unsupervised Learning	202
17.1	Association Rules/Market Basket Analysis	202
17.1.1	The "Apriori Algorithm" and Use of its Output	204
17.2	Clustering	206
17.2.1	Partitioning Methods ("Centroid"-Based Methods)	207
17.2.2	Hierarchical Methods	208
17.2.3	(Mixture) Model-Based Methods	210
17.2.4	Biclustering	211
17.2.5	Self-Organizing Maps	214
17.3	Multi-Dimensional Scaling	218
17.4	More on Principal Components and Related Ideas	220
17.4.1	"Sparse" Principal Components	220
17.4.2	Non-negative Matrix Factorization	221
17.4.3	Archetypal Analysis	222
17.4.4	Independent Component Analysis	222
17.4.5	Principal Curves and Surfaces	225
17.5	(Original) Google PageRanks	228
VI	Miscellanea	230
18	Graphs as Representing Independence Relationships in Multi-variate Distributions	230
18.1	Some Considerations for Directed Graphical Models	231
18.2	Some Considerations for Undirected Graphical Models	233
18.2.1	Restricted Boltzmann Machines	235
19	Special Bayes Methods for Statistical Learning	240
19.1	Relevance Vector Machines	240
19.2	Dirichlet and Data-Derived Priors for Prediction Based on Normal Mixture Models	242
19.3	Bayes Mixture Analyses for Binary Vectors	244
VII	Appendices	245
A	Exercises	245
A.1	Section 1.2 Exercises	245
A.2	Section 1.3 Exercises	247
A.3	Section 1.4 Exercises	261
A.4	Section 1.5 Exercises	267
A.5	Section 1.6 Exercises	270
A.6	Section 2.1 Exercises	271
A.7	Section 2.2 Exercises	272
A.8	Section 2.3 Exercises	274
A.9	Section 2.4 Exercises	276

A.10 Section 3.1 Exercises	281
A.11 Section 3.2 Exercises	285
A.12 Section 4.1 Exercises	287
A.13 Section 4.2 Exercises	289
A.14 Section 4.3 Exercises	290
A.15 Section 5.1 Exercises	291
A.16 Section 5.2 Exercises	292
A.17 Section 5.3 Exercises	293
A.18 Section 6.1 Exercises	293
A.19 Section 6.2 Exercises	298
A.20 Section 7.1 Exercises	299
A.21 Section 8.1 Exercises	299
A.22 Section 8.2 Exercises	302
A.23 Section 9.1 Exercises	303
A.24 Section 10.1 Exercises	306
A.25 Section 10.2 Exercises	307
A.26 Section 11.1 Exercises	308
A.27 Section 11.2 Exercises	310
A.28 Section 11.4 Exercises	311
A.29 Section 12.1 Exercises	315
A.30 Section 12.2 Exercises	317
A.31 Section 13.1 Exercises	319
A.32 Section 13.2 Exercises	319
A.33 Section 13.3 Exercises	320
A.34 Section 14 Exercises	323
A.35 Section 15.2 Exercises	323
A.36 Section 15.3 Exercises	324
A.37 Section 15.4 Exercises	325
A.38 Section 17.1 Exercises	326
A.39 Section 17.2 Exercises	326
A.40 Section 17.3 Exercises	328
A.41 Section 18.2.1 Exercises	329
A.42 "General/Comprehensive" Exercises	329

Part I

Introduction, Generalities, and Some Background Material

1 Overview/Context

1.1 Notation and Terminology

These notes are about "statistics for 'big data'" (AKA "machine learning" and "data analytics"). We begin with the standard statistical notation and set-up where one has data from N cases on p or $p + 1$ variables, x_1, x_2, \dots, x_p and possibly y portrayed below:

		Variables				
		x_{11}	x_{12}	\cdots	x_{1p}	y_1
		x_{21}	x_{22}	\cdots	x_{2p}	y_2
Cases		\vdots	\vdots	\ddots	\vdots	\vdots
		x_{N1}	x_{N2}	\cdots	x_{Np}	y_N

In statistical machine learning, this dataset is typically called the **training** dataset and we'll call it \mathbf{T} . Variables are often referred to as **features**, and cases are sometimes called **instances**. We'll use standard matrix (and linear models) notation, beginning with

$$\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{ip})'$$

for the case/row i set of x values (in column vector form unless otherwise indicated) and

$$\mathbf{X}_{N \times p} = \begin{pmatrix} \mathbf{x}'_1 \\ \mathbf{x}'_2 \\ \vdots \\ \mathbf{x}'_N \end{pmatrix}, \mathbf{Y}_{N \times 1} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}, \text{ and } \mathbf{T} = (\mathbf{X}, \mathbf{Y})$$

for the training data.

As in all of statistics, the basic objective is identifying, describing, and enabling the practical use of simple (low-dimensional/low-order) structure represented in the $N \times p$ or $N \times (p + 1)$ data array. Most "classical" statistical methods are implicitly aimed at situations where

- both N and p are small (data are scarce), *and*
- quantifications of the level of information the data provide about parameters of a probability model are central.

Here we treat cases where at least one of N or p can be large and there is little fundamental interest in model parameters or exactly how much we know about them.

The ambivalence toward making statements concerning parameters of any probability models employed (including those that would describe the amount of variation in observables the model generates) is a fundamental difference between a machine learning point of view and that common in basic graduate statistics courses. This posture is perhaps sensible enough, as careful examination of a large training set will usually show that standard (tractable) probability models are highly imperfect descriptions of complex situations.

Standard versions of problems addressed here are:

- **supervised learning** problems¹, where there is a response/output variable or **target**, y , and the problem is one of finding a function of p inputs \mathbf{x} , $f(\mathbf{x})$, that approximates y . When the form of f depends on the training set, we'll write $\hat{y} = \hat{f}(\mathbf{x})$. Where y is a measured/continuous variable the problem is typically called **prediction**. Where y takes values in a finite set like $\{1, 2, \dots, K\}$, the problem is typically called **classification** (or sometimes **pattern recognition**). In these notes, we will on occasion wish to refer simultaneously to both standard forms of supervised learning and may then treat the word "prediction" as including both cases.
- **unsupervised learning** problems, where there is no response variable. The general objective here is then to identify relationships among the p variables x or commonalities in segments of the N cases, i.e. interpretable low-order structure in the data. Standard versions of this are **clustering**, **principal components analysis**, and **multi-dimensional scaling**.

1.2 What is New Here (Particularly in Prediction)?

Reasonable questions here are "What is the big deal?" and "What new issues arise in 'statistics for big data'?"

Where N and/or p are large, limitations on computing time or computer memory can make straightforward implementation of standard methods impractical or even impossible. Sometimes more clever implementations (for example employing parallelization or use of specialized hardware) make application of standard methods feasible. (These are matters that won't be much considered in these notes.) At other times, new methods need to be developed.

1.2.1 Matching Complexity to Training Set Information Content

Where N is big and p is small, standard statistical prediction methods (like multiple linear regression) will produce precisely fit but relatively crude predictors ... whose forms, while perhaps adequate as first approximations to a real relationship between \mathbf{x} and y (and about all that *can* be fit based on a

¹In this context the input variables \mathbf{x} are "**covariates**" in standard statistical parlance.

small dataset), fail to really make full use of the available information. There is the possibility of either increasing " p " by (implicitly or explicitly) building additional features from existing ones and/or simply using more sophisticated and flexible forms for prediction (that go beyond, for example, the basic linear form in the input variables of multiple linear regression). But there is also the potential to "over-do" and effectively make p too large or the predictor too flexible. **One must somehow match predictor complexity to the real information content of a (large) training set.** It is this need and the challenge it represents that makes the area interesting and important.

1.2.2 The "Curse of Dimensionality"

If p is at all big, \mathfrak{R}^p is "huge" and our intuition about how many cases would be required to "fill up" even an intuitively small part of p -space is very poor. Essentially *any* dataset with large p is necessarily "sparse." There are many ways of framing this inescapable sparsity. Some simple ones involve facts about uniform distributions on the unit ball in \mathfrak{R}^p

$$\{\mathbf{x} \in \mathfrak{R}^p \mid \|\mathbf{x}\| \leq 1\}$$

and on a unit cube centered at the origin

$$[-.5, .5]^p$$

For one thing, "most" of these distributions are very near the surface of the solids. The cube $[-r, r]^p$ capturing (for example) half the volume of the cube (half the probability mass of the distribution) has

$$r = .5 (.5)^{1/p}$$

which converges to .5 as p increases. Essentially the same story holds for the uniform distribution on the unit ball. The radius capturing half the probability mass has

$$r = (.5)^{1/p}$$

which converges to 1 as p increases. Points uniformly distributed in these regions are mostly near the surface or boundary of the spaces.

Another interesting calculation concerns how large a sample must be in order for points generated from the uniform distribution on the ball or cube in an iid fashion to tend to "pile up" anywhere. Consider the problem of describing the distance from the origin to the closest of N points drawn iid uniformly from the p -dimensional unit ball. With

R = the distance from the origin to a single random point ,

R has cdf

$$F(r) = \begin{cases} 0 & r < 0 \\ r^p & 0 \leq r \leq 1 \\ 1 & r > 1 \end{cases}$$

So if R_1, R_2, \dots, R_N are iid with this distribution, $M = \min\{R_1, R_2, \dots, R_N\}$ has cdf

$$F_M(m) = \begin{cases} 0 & m < 0 \\ 1 - (1 - m^p)^N & 0 \leq m \leq 1 \\ 1 & m > 1 \end{cases}$$

This distribution has, for example, median

$$F_M^{-1}(.5) = \left(1 - \left(\frac{1}{2}\right)^{1/N}\right)^{1/p}$$

For, say, $p = 100$ and $N = 10^6$, the median of the distribution of M is .87.

In retrospect, it's not really all that hard to understand that even "large" sets of points in \mathfrak{R}^p must be sparse. After all, it's perfectly possible for p -vectors \mathbf{u} and \mathbf{v} to agree perfectly on all but one coordinate and be far apart in p -space. There simply are a lot of ways for two p -vectors to differ!

In addition to these kinds of considerations of sparsity, there is the fact that the potential complexity of functions of p variables explodes exponentially in p . CFZ point this out and go on to note that for large p , all datasets exhibit multicollinearity (or its generalization to non-parametric fitting) and its accompanying problems of reliable fitting and extrapolation. These and related issues together constitute what is often called the **curse of dimensionality**.

The curse implies that even "large" N doesn't save one and somehow make practical large- p prediction only a trivial application of standard parametric or non-parametric regression methodology. And when p is large, it is essentially guaranteed that if one uses a method that is "too" flexible in terms of the relationships between \mathbf{x} and y that it permits, one will be found, real/fundamental/reproducible or not. That is, the (common for large p) possibility that a dataset is (sparse and) not really adequate to support the use of a (flexible) supervised statistical learning method can easily lead to **overfitting**. This is the presence of what appears to be a strong pattern in a (sparse) training set that generalizes/extrapolates poorly to cases outside the training set.

In light of the foregoing, one standard way of choosing among various "big data" statistical procedures for a given dataset is to define both 1) a reliable measure of estimated/predicted performance (like an estimated prediction mean square error) and 2) a measure of complexity (like an "effective number of fitted parameters") for a predictor. Then one attempts to optimize (by choice of complexity measure) the predicted performance. In light of the overfitting issue, the method predicting performance almost always employs some form of **"holdout" sample**, whereby performance is evaluated using data *not* employed in fitting/predictor development.²

²This approach potentially addresses the detection of both overfitting and "model bias" (where a fitted form is simply not adequate to represent the relationship between input variables and a target).

1.3 Some Initial Generalities About Prediction

1.3.1 Representing What is Known: Creating a Training Set for Prediction

We began exposition with an $N \times (p + 1)$ data matrix conceptually already in hand. It is important to say that in real predictive analytics problems, the reduction of all information available and potentially relevant to explaining y to values of p predictor variables³ (that encode relevant "features" of the N cases) is an essential and highly critical activity. If one defines good features/variables (ones that effectively and parsimoniously represent the N cases), then sound statistical methodology has a chance of being practically helpful. Poor initial choice of features limits how well one can hope to do in prediction.

This is particularly important to bear in mind where information from many disparate databases or sources is used to create the training set/data matrix \mathbf{T} available for statistical analysis. In this way, in many applications of modern data analytics the hard work begins substantially *before* the formal technical subjects addressed in these notes come into play, and the quality of the work in those initial steps is critical to ultimate success. **All that follows in these notes takes the particular form of training set adopted by a data analyst as given, and that choice governs and limits what is possible in terms of effective prediction.**

We should also note that in a typical analytics problem, variables represented by the columns of a data matrix are in different units and often represent conceptually different kinds of quantities (e.g., one might represent a voltage while another represents a distance and another represents a temperature). In some kinds of analyses this is completely natural and causes no logical problems. But in others (particularly ones based on inner products of data vectors or distances between them and/or where sizes of multipliers of particular variables in a linear combination of those variables are important) one gets fundamentally different results depending upon the scales used.

One surely doesn't want to be in the position of having ultimate predictions depend upon whether a distance (represented by a coordinate of \mathbf{x}) is expressed in km or in nm. And the whole notion of the \mathfrak{R}^2 distance between two data vectors where the first coordinate of each is a voltage and the second is a temperature seems less than attractive. (What is $\sqrt{(3 \text{ kV})^2 + (2^\circ \text{ K})^2}$ supposed to mean?)

A sensible approach to eliminating logical difficulties that arise in using methods where scaling/units of variables matters, is to standardize predictors x (and center any quantitative response variable, y) before beginning analysis. That is, if a raw feature x has in the training set a sample standard deviation⁴

³This is at least one common meaning of the term "data mining."

⁴While it doesn't really matter which one uses, the " N " divisor in place of the " $N - 1$ " divisor seems slightly simpler as it makes the columns have \mathfrak{R}^N norm \sqrt{N} as opposed to norm $\sqrt{N - 1}$.

s_x and a sample mean \bar{x} , one replaces it with a feature

$$x' \equiv \frac{x - \bar{x}}{s_x}$$

(thereby making all features unit-less). Conclusions about standardized input x' and centered response $y' = y - \bar{y}$ then translate naturally to conclusions about the raw variables via

$$x = s_x \cdot x' + \bar{x} \quad \text{and} \quad y = y' + \bar{y}$$

1.3.2 Theoretically Optimal (Unrealizable) Predictors

In the context of supervised learning and the objective of choosing $f(\mathbf{x})$ to track y , suppose that P is a $((p+1)$ -dimensional) distribution for (\mathbf{x}, y) and $L(\hat{y}, y) \geq 0$ is a loss function for penalizing prediction/classification \hat{y} when y holds. Let "E" be the P expectation operator. Unless specifically noted to the contrary, all expectations refer to distributions and conditional distributions derived from P . In general, if we need to remind ourselves what variables are being treated as random in probability, expectation, conditional probability, or conditional expectation computations, we will superscript E or P with their names.) Write $E[\cdot|\mathbf{x}]$ for conditional expectation and $\text{Var}[\cdot|\mathbf{x}]$ for conditional variance (based on the conditional distribution of $y|\mathbf{x}$) derived from P .

As a thought experiment (not yet as anything based on the training data) consider choosing a functional form f (NOT yet \hat{f}) to minimize risk (or "prediction error")

$$EL(f(\mathbf{x}), y)$$

In theory (given P) this is "easy." One writes the expectation in iterated fashion,

$$EL(f(\mathbf{x}), y) = EE[L(f(\mathbf{x}), y) | \mathbf{x}]$$

and notes that an optimal $f(\mathbf{x})$ is thus

$$f(\mathbf{x}) = \arg \min_a E[L(a, y) | \mathbf{x}] \tag{1}$$

the action/prediction that minimizes conditional (on the value of \mathbf{x}) expected (over y) loss.

SEL In the simple case of squared error loss, i.e. where

$$L(\hat{y}, y) = (\hat{y} - y)^2$$

an optimal f in display (1) is then well-known to be

$$f(\mathbf{x}) = E[y|\mathbf{x}]$$

the conditional mean of $y|\mathbf{x}$.

Classification In a classification context, where y takes values in $\mathcal{G} = \{1, 2, \dots, K\}$ (or, completely equivalently, $\mathcal{G} = \{0, 1, \dots, K - 1\}$), one might use the (0-1) loss function

$$L(\hat{y}, y) = I[\hat{y} \neq y]$$

An optimal f corresponding to form (1) is then

$$\begin{aligned} f(\mathbf{x}) &= \arg \min_a \sum_{v \neq a} P[y = v | \mathbf{x}] \\ &= \arg \max_a P[y = a | \mathbf{x}] \\ &= \arg \max_a P[y = a] p(\mathbf{x} | a) \end{aligned} \quad (2)$$

(where $p(\mathbf{x} | y)$ is a density for the class-conditional distribution of $\mathbf{x} | y$).

A simple generalization of 0-1 loss is one that for different values of y charges potentially different losses $l_y \geq 0$ when $\hat{y} \neq y$, that is,

$$L(\hat{y}, y) = l_y I[\hat{y} \neq y]$$

Essentially the same argument as above implies that an optimal f for this possibly asymmetric loss is

$$\begin{aligned} f(\mathbf{x}) &= \arg \min_a \sum_{v \neq a} l_v P[y = v | \mathbf{x}] \\ &= \arg \max_a l_a P[y = a | \mathbf{x}] \\ &= \arg \max_a l_a P[y = a] p(\mathbf{x} | a) \end{aligned}$$

Another Problem in Classification Models In a classification model as immediately above, one might have in mind assessment of the set of likelihoods that $y = k$ based on \mathbf{x} . That would call for the making of a K -dimensional predictor $\hat{\mathbf{y}}$ and appropriate definition of a loss. One simple possible loss is a sum of squared errors

$$L(\hat{\mathbf{y}}, y) = \sum_{k=1}^K (\hat{y}_k - I[y = k])^2$$

for which it is easy to show that an optimal vector predictor is

$$\mathbf{f}(\mathbf{x}) = (P[y = 1 | \mathbf{x}], P[y = 2 | \mathbf{x}], \dots, P[y = K | \mathbf{x}]) \quad (3)$$

For many purposes (see, e.g., Section 8.2) a more appropriate loss is the "cross-entropy loss"

$$L(\hat{\mathbf{y}}, y) = - \sum_{k=1}^K I[y = k] \ln \hat{y}_k \quad (4)$$

What is perhaps not immediately obvious is that a Lagrange multiplier argument shows that subject to the constraint that the \hat{y}_k are positive and sum to 1, the vector predictor (3) is also optimal for cross-entropy loss (4).

1.3.3 Nearest Neighbor Rules

One idea that create a spectrum of predictor flexibilities (including extremely high ones) is to operate completely non-parametrically and to think that if N is "big enough," something like

$$\frac{1}{\# \text{ of } \mathbf{x}_i = \mathbf{x}} \sum_{\substack{i \text{ with} \\ \mathbf{x}_i = \mathbf{x}}} y_i \quad (5)$$

might work as an approximation to $E[y|\mathbf{x}]$ in SEL prediction problems and

$$\frac{1}{\# \text{ of } \mathbf{x}_i = \mathbf{x}} \sum_{\substack{i \text{ with} \\ \mathbf{x}_i = \mathbf{x}}} I[y_i = a] \quad (6)$$

might work as an approximation for $P[y = a|\mathbf{x}]$ in a K -class classification model. But almost always (unless N is huge and the distribution of \mathbf{x} is discrete) the number of $\mathbf{x}_i = \mathbf{x}$ is 0 or at most 1 (and absolutely no extrapolation beyond the set of training inputs is possible). So some modification is typically required. The condition

$$\mathbf{x}_i = \mathbf{x}$$

might be replaced with

$$\mathbf{x}_i \approx \mathbf{x}$$

in expression (5) and/or (6).

One form of this is to first define for each \mathbf{x} the " k -neighborhood"

$$n_k(\mathbf{x}) = \text{the set of } k \text{ inputs } \mathbf{x}_i \text{ in the training set closest to } \mathbf{x} \text{ in } \mathfrak{R}^p$$

A k -nearest neighbor (k -nn) approximation to $E[y|\mathbf{x}]$ is then

$$m(\mathbf{x}) = \frac{1}{k} \sum_{\substack{i \text{ with} \\ \mathbf{x}_i \in n_k(\mathbf{x})}} y_i$$

suggesting the SEL prediction rule

$$\hat{f}(\mathbf{x}) = m(\mathbf{x})$$

Similarly, a k -nearest neighbor approximation to an optimal 0-1 loss classification rule in a K -class classification model is

$$\hat{f}(\mathbf{x}) = \arg \max_a \sum_{\substack{i \text{ with} \\ \mathbf{x}_i \in n_k(\mathbf{x})}} I[y_i = a]$$

One might hope that upon allowing k to increase with N (provided that P is not too bizarre—one is counting, for example, on the continuity of $E[y|\mathbf{x}]$ in \mathbf{x}) these could be effective predictors. They are surely (for small k) highly flexible

predictors and they and things like them often fail to be effective because of the curse of dimensionality. (In high dimensions, k -neighborhoods are almost always huge in terms of their extent. There are simply too many ways that a pair of training inputs \mathbf{x}_i can differ.

It is worth noting that

$$m^a(\mathbf{x}) = \frac{1}{k} \sum_{\substack{i \text{ with} \\ \mathbf{x}_i \in n_k(\mathbf{x})}} I[y_i = a]$$

is a k -nearest neighbor approximation to

$$E[I[y = a] | \mathbf{x}] = P[y = a | \mathbf{x}]$$

in a K -class model, and for some purposes knowing this is more useful than knowing the 0-1 loss k -nn classification rule.

Ultimately, one should view the k -nn idea as an important, almost deceptively simple, and highly useful one. k -nn rules are approximately optimal predictors (for both SEL and 0-1 loss problems) that span a full spectrum of complexities/flexibilities specified by the simple parameter k (the neighborhood size). Whether or not they can be effective in a given application depends upon the size of p and N and the extent to which there is some useful structure latent in the distribution of \mathbf{x} s in the input space (mitigating the effects of the curse of dimensionality).

1.3.4 General Decomposition of the Expected Prediction Loss for \hat{f}

Now suppose that the training data (\mathbf{x}_i, y_i) for $i = 1, \dots, N$ are iid according to P , independent of a single (\mathbf{x}, y) that is also P distributed.⁵ Write $E^{\mathbf{T}}$ for averaging with respect to P^N (i.e. for averaging out over the training data), and $E^{(\mathbf{x}, y)}$ for averaging with respect to the distribution P of (\mathbf{x}, y) .

For \hat{f} a predictor based on the training data \mathbf{T} (a function of both \mathbf{x} and \mathbf{T}), a measure of average effectiveness of \hat{f} is the prediction error⁶

$$\text{Err} \equiv E^{\mathbf{T}} E^{(\mathbf{x}, y)} \left[L(\hat{f}(\mathbf{x}), y) \right] \quad (7)$$

If $f(\mathbf{x})$ is a theoretically optimal predictor of y under loss L and joint distribution P , training set \mathbf{T} is used to select a function (say $g_{\mathbf{T}}$) from a class of functions $\mathcal{S} = \{g\}$ having expected loss $E^{(\mathbf{x}, y)} L(g(\mathbf{x}), y) < \infty$, and ultimately one uses as a predictor

$$\hat{f}(\mathbf{x}) = g_{\mathbf{T}}(\mathbf{x}) \quad ,$$

the situation is as in the cartoon in Figure 1, where g^* is a minimizer of $E^{(\mathbf{x}, y)} L(g(\mathbf{x}), y)$ across $g \in \mathcal{S}$.

⁵We will typically abuse notation and write (\mathbf{x}, y) instead of (\mathbf{x}', y) despite the fact that by convention \mathbf{x} is a column vector.

⁶This quantity is sometimes called the "test error" and "generalization error" and these names will also be used in our exposition and problem sets.

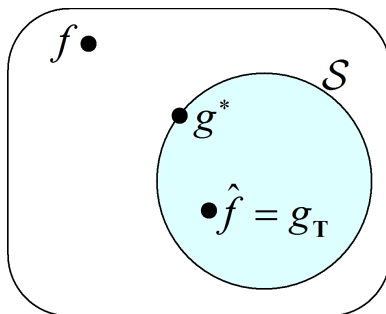


Figure 1: Optimal, Restricted Optimal, and Fitted Predictors

The optimal $f(\mathbf{x})$ is potentially (likely) outside of \mathcal{S} . The "closest" one can get to it inside of \mathcal{S} is g^* , and lacking full knowledge of P one can only approximate this best element of \mathcal{S} by the random (as the choice depends upon the training set \mathbf{T}) $\hat{f} = g_{\mathbf{T}}$ (that is no better than g^* for *any* training set!).

So, since here

$$\text{Err} = \mathbb{E}^{\mathbf{T}} \mathbb{E}^{(\mathbf{x}, y)} L(\hat{f}(\mathbf{x}), y) = \mathbb{E}^{\mathbf{T}} \mathbb{E}^{(\mathbf{x}, y)} L(g_{\mathbf{T}}(\mathbf{x}), y)$$

we have

$$\begin{aligned} \text{Err} = & \mathbb{E}^{(\mathbf{x}, y)} L(f(\mathbf{x}), y) + \left(\mathbb{E}^{(\mathbf{x}, y)} L(g^*(\mathbf{x}), y) - \mathbb{E}^{(\mathbf{x}, y)} L(f(\mathbf{x}), y) \right) \\ & + \left(\mathbb{E}^{\mathbf{T}} \mathbb{E}^{(\mathbf{x}, y)} L(g_{\mathbf{T}}(\mathbf{x}), y) - \mathbb{E}^{(\mathbf{x}, y)} L(g^*(\mathbf{x}), y) \right) \end{aligned} \quad (8)$$

This says that Err for the training-set-dependent predictor is the sum of three terms. The first is the minimum possible error. The second is the *non-negative* difference between the best that is possible using a predictor constrained to be an element of \mathcal{S} and the absolute best that is possible. The third is the *non-negative* difference between Err (that involves averaging over the training-set-directed random choices of elements from \mathcal{S} , none of which can have average loss over (\mathbf{x}, y) better than that of g^*) and the best that is possible using a predictor constrained to be an element of \mathcal{S} (namely the average loss of $g^*(\mathbf{x})$). So relationship (8) might be rewritten as

$$\begin{aligned} \text{Err} = & \text{minimum expected loss possible} + \text{modeling penalty} \\ & + \text{fitting penalty} \end{aligned}$$

Err can be inflated because \mathcal{S} is too small (inducing **model bias**) or because the sample size and/or fitting method are inadequate to make $g_{\mathbf{T}}$ consistently approximate g^* .

1.3.5 A More Detailed Decomposition for Err in SEL Prediction and Variance-Bias Trade-off

In the context of squared error loss, a more detailed decomposition of Err provides additional insight into the difficulty faced in building effective predictors.

Note that a measure of the effectiveness of the predictor \hat{f} at \mathbf{x} (under squared error loss) is what we might call

$$\text{Err}(\mathbf{x}) \equiv \mathbb{E}^{\mathbf{T}} \mathbb{E} \left[\left(\hat{f}(\mathbf{x}) - y \right)^2 \mid \mathbf{x} \right] \quad (9)$$

For some purposes, other conditional versions of Err might be useful and appropriate. For example

$$\text{Err}_{\mathbf{T}} \equiv \mathbb{E}^{(\mathbf{x};y)} \left(\hat{f}(\mathbf{x}) - y \right)^2$$

is another kind of prediction or test error (that is a function of the training data). (What one would surely like—but surely cannot have—is a guarantee that $\text{Err}_{\mathbf{T}}$ is small *uniformly in \mathbf{T}* .) Note that in these notations what we have called

$$\text{Err} \equiv \mathbb{E}^{\mathbf{x}} \text{Err}(\mathbf{x}) = \mathbb{E}^{\mathbf{T}} \text{Err}_{\mathbf{T}} \quad (10)$$

is a *number*, an expected squared difference between target and prediction.

In any case, a useful decomposition of $\text{Err}(\mathbf{x})$ in display (9) is

$$\begin{aligned} \text{Err}(\mathbf{x}) &= \mathbb{E}^{\mathbf{T}} \left\{ \left(\hat{f}(\mathbf{x}) - \mathbb{E}[y|\mathbf{x}] \right)^2 + \mathbb{E} \left[(y - \mathbb{E}[y|\mathbf{x}])^2 \mid \mathbf{x} \right] \right\} \\ &= \mathbb{E}^{\mathbf{T}} \left\{ \left(\hat{f}(\mathbf{x}) - \mathbb{E}^{\mathbf{T}} \hat{f}(\mathbf{x}) \right)^2 + \left(\mathbb{E}^{\mathbf{T}} \hat{f}(\mathbf{x}) - \mathbb{E}[y|\mathbf{x}] \right)^2 \right\} + \text{Var}[y|\mathbf{x}] \\ &= \text{Var}^{\mathbf{T}} \left(\hat{f}(\mathbf{x}) \right) + \left(\mathbb{E}^{\mathbf{T}} \hat{f}(\mathbf{x}) - \mathbb{E}[y|\mathbf{x}] \right)^2 + \text{Var}[y|\mathbf{x}] \end{aligned} \quad (11)$$

The first quantity in this decomposition, $\text{Var}^{\mathbf{T}} \left(\hat{f}(\mathbf{x}) \right)$, is the variance of the prediction at \mathbf{x} . The second term, $\left(\mathbb{E}^{\mathbf{T}} \hat{f}(\mathbf{x}) - \mathbb{E}[y|\mathbf{x}] \right)^2$, is a kind of squared bias of prediction at \mathbf{x} . And $\text{Var}[y|\mathbf{x}]$ is an unavoidable variance in outputs at \mathbf{x} . Highly flexible prediction forms may give small prediction biases at the expense of large prediction variances. One may need to balance the two off against each other when looking for a good predictor.

Now from expressions (10) and (11)

$$\begin{aligned} \text{Err} &= \mathbb{E}^{\mathbf{x}} \text{Err}(\mathbf{x}) \\ &= \mathbb{E}^{\mathbf{x}} \text{Var}^{\mathbf{T}} \left(\hat{f}(\mathbf{x}) \right) + \mathbb{E}^{\mathbf{x}} \left(\mathbb{E}^{\mathbf{T}} \hat{f}(\mathbf{x}) - \mathbb{E}[y|\mathbf{x}] \right)^2 + \mathbb{E}^{\mathbf{x}} \text{Var}[y|\mathbf{x}] \end{aligned} \quad (12)$$

The first term on the right here is the average (according to the marginal of \mathbf{x}) of the prediction variance at \mathbf{x} . The second is the average squared prediction

bias. And the third is the average conditional variance of y (and is not under the control of the analyst choosing $\hat{f}(\mathbf{x})$). Consider a further decomposition of the second term.

Suppose that \mathbf{T} is used to select a function (say $g_{\mathbf{T}}$) from some linear subspace, say $\mathcal{S} = \{g\}$, of the space functions h with $\mathbb{E}^{\mathbf{x}}(h(\mathbf{x}))^2 < \infty$, and that ultimately one uses as a predictor

$$\hat{f}(\mathbf{x}) = g_{\mathbf{T}}(\mathbf{x})$$

Since linear subspaces are convex

$$g^{**} \equiv \mathbb{E}^{\mathbf{T}} g_{\mathbf{T}} = \mathbb{E}^{\mathbf{T}} \hat{f} \in \mathcal{S}$$

Further, suppose that

$$g^* \equiv \arg \min_{g \in \mathcal{S}} \mathbb{E}^{\mathbf{x}} (g(\mathbf{x}) - \mathbb{E}[y|\mathbf{x}])^2$$

is the projection of (the function of \mathbf{x}) $\mathbb{E}[y|\mathbf{x}]$ onto the space \mathcal{S} . Then write

$$h^*(\mathbf{x}) = \mathbb{E}[y|\mathbf{x}] - g^*(\mathbf{x})$$

so that

$$\mathbb{E}[y|\mathbf{x}] = g^*(\mathbf{x}) + h^*(\mathbf{x})$$

Then, it's a consequence of the facts that $\mathbb{E}^{\mathbf{x}}(h^*(\mathbf{x})g(\mathbf{x})) = 0$ for all $g \in \mathcal{S}$ and therefore that $\mathbb{E}^{\mathbf{x}}(h^*(\mathbf{x})g^*(\mathbf{x})) = 0$ and $\mathbb{E}^{\mathbf{x}}(h^*(\mathbf{x})g^{**}(\mathbf{x})) = 0$, that

$$\begin{aligned} \mathbb{E}^{\mathbf{x}} \left(\mathbb{E}^{\mathbf{T}} \hat{f}(\mathbf{x}) - \mathbb{E}[y|\mathbf{x}] \right)^2 &= \mathbb{E}^{\mathbf{x}} (g^{**}(\mathbf{x}) - (g^*(\mathbf{x}) + h^*(\mathbf{x})))^2 \\ &= \mathbb{E}^{\mathbf{x}} (g^{**}(\mathbf{x}) - g^*(\mathbf{x}))^2 + \mathbb{E}^{\mathbf{x}} (h^*(\mathbf{x}))^2 \\ &\quad - 2\mathbb{E}^{\mathbf{x}} ((g^{**}(\mathbf{x}) - g^*(\mathbf{x})) h^*(\mathbf{x})) \\ &= \mathbb{E}^{\mathbf{x}} \left(\mathbb{E}^{\mathbf{T}} \hat{f}(\mathbf{x}) - g^*(\mathbf{x}) \right)^2 + \mathbb{E}^{\mathbf{x}} (\mathbb{E}[y|\mathbf{x}] - g^*(\mathbf{x}))^2 \quad (13) \end{aligned}$$

The first term on the right in the last line of display (13) is an average squared fitting bias, measuring how well the average (over \mathbf{T}) predictor function approximates the element of \mathcal{S} that best approximates the conditional mean function. This is a measure of how appropriately the training data are used to pick out elements of \mathcal{S} . The second term on the right is an average squared model bias, measuring how well it is possible to approximate the conditional mean function $\mathbb{E}[y|\mathbf{x}]$ by an element of \mathcal{S} . This is controlled by the size of \mathcal{S} , or effectively the flexibility allowed in the form of \hat{f} . Average squared prediction bias can thus be large because the form fit is not flexible enough, or because a poor fitting method is employed.

Then using expressions (12) and (13)

$$\begin{aligned} \text{Err} &= \mathbb{E}^{\mathbf{x}} \text{Var}[y|\mathbf{x}] + \mathbb{E}^{\mathbf{x}} (\mathbb{E}[y|\mathbf{x}] - g^*(\mathbf{x}))^2 \\ &\quad + \mathbb{E}^{\mathbf{x}} \left(\mathbb{E}^{\mathbf{T}} \hat{f}(\mathbf{x}) - g^*(\mathbf{x}) \right)^2 + \mathbb{E}^{\mathbf{x}} \text{Var}^{\mathbf{T}}(\hat{f}(\mathbf{x})) \end{aligned}$$

So this SEL decomposition of Err is related to the general one in display (8) in that

$$\begin{aligned} \text{minimum expected loss possible} &= \text{expected (across } \mathbf{x} \text{) response variance} \\ &= \mathbf{E}^{\mathbf{x}} \text{Var} [y|\mathbf{x}], \end{aligned}$$

$$\begin{aligned} \text{modeling penalty} &= \text{expected (across } \mathbf{x} \text{) squared model bias} \\ &= \mathbf{E}^{\mathbf{x}} (\mathbf{E} [y|\mathbf{x}] - g^*(\mathbf{x}))^2, \end{aligned}$$

and

$$\begin{aligned} \text{fitting penalty} &= \left(\begin{array}{c} \text{expected (across } \mathbf{x} \text{)} \\ \text{squared fitting bias} \end{array} \right) + \left(\begin{array}{c} \text{expected (across } \mathbf{x} \text{)} \\ \text{prediction variance} \end{array} \right) \\ &= \mathbf{E}^{\mathbf{x}} \left(\mathbf{E}^T \hat{f}(\mathbf{x}) - g^*(\mathbf{x}) \right)^2 + \mathbf{E}^{\mathbf{x}} \text{Var}^T \left(\hat{f}(\mathbf{x}) \right) \end{aligned}$$

The facts that

1. what is under the control of a data analyst, namely the modeling and fitting penalties, has elements of both bias and variance and
2. complex predictors tend to have low bias and high variance in comparison to simple ones

leads to the necessity of balancing these elements in predictor development and the so-called **variance-bias trade-off**. Once more, in qualitative terms, **it is the matching of predictor complexity to real information content of a training set that is at issue here.**

1.3.6 Approximating Err and Cross-Validation

In rough terms, standard methods of constructing predictors all have associated "complexity parameters" (like k for k -nearest neighbor methods, numbers and types of features/basis functions or "ridge parameters" used in regression methods, and penalty weights/band-widths/neighborhood sizes applied in smoothing methods) that are at the choice of a user. Depending upon the choice of complexity, one gets more or less flexibility in the form \hat{f} . If a choice of complexity doesn't allow enough flexibility in the form of a predictor, underfit occurs and there is large bias in prediction. On the other hand, if the choice allows too much flexibility, bias may be reduced, but the price typically paid is large variance of prediction and overfit. **It is a theme that runs through this material that complexity must be chosen in a way that balances variance and bias for the particular combination of N and p and general circumstance one faces.** That choice of predictor complexity of course depends upon reliable means of assessing (the unknown theoretical test error) Err in display (7).

The most obvious/elementary means of approximating Err is the so-called "training error"

$$\overline{\text{err}} = \frac{1}{N} \sum_{i=1}^N L(\hat{f}(\mathbf{x}_i), y_i) \quad (14)$$

The problem is that $\overline{\text{err}}$ is no good estimator of Err (or any other sensible quantification of predictor performance). It typically decreases with increased complexity (without an increase for large complexity), and fails to reliably indicate performance outside the training sample. The situation is like that portrayed in Figure 2.

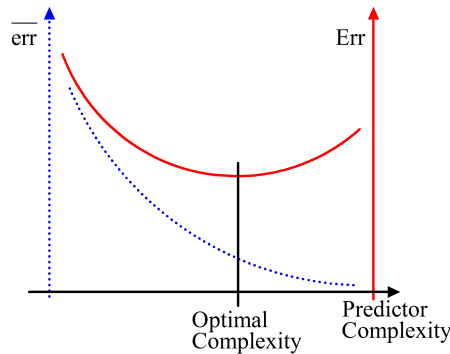


Figure 2: Cartoon Portraying Err and $\overline{\text{err}}$ as Functions of Predictor Complexity

The fundamental point here is that one cannot both "fit" and "test" on the same dataset and arrive at a reliable assessment of predictor efficacy. Behaving in such manner will almost always suggest use of a predictor that is too complex and has a relatively large "test error" Err.

The existing practical options for evaluating likely performance of a predictor (and guiding choice of appropriate complexity) then include the following.

1. One might employ some function of $\overline{\text{err}}$ that is a better indicator of likely predictor performance, like Mallows' C_p , "AIC," and "BIC."
2. In genuinely large N contexts, one might hold back some random sample of the training data to serve as a "test set," fit to produce \hat{f} on the remainder, and use

$$\frac{1}{\text{size of the test set}} \sum_{z \in \text{the test set}} L(\hat{f}(\mathbf{x}_z), y_z)$$

to indicate likely predictor performance.

3. One might employ sample re-use methods to estimate Err and guide choice of complexity. Cross-validation and bootstrap ideas are used here.

We'll say more about these possibilities later, but here describe the most important of them, so-called **cross-validation**. K -fold cross-validation consists of

1. randomly breaking the training set into K disjoint roughly equal-sized pieces ("**folds**"), say $\mathbf{T}_1, \mathbf{T}_2, \dots, \mathbf{T}_K$,
2. training on each of the reduced training sets $\mathbf{T} - \mathbf{T}_k$ (that we will call corresponding "**remainders**") to produce K predictors \hat{f}^k ,
3. letting $k(i)$ be the index of the fold \mathbf{T}_k containing training case i , and computing the cross-validation error

$$CV(\hat{f}) = \frac{1}{N} \sum_{i=1}^N L(\hat{f}^{k(i)}(\mathbf{x}_i), y_i) \quad (15)$$

that one hopes approximates Err .

(This is roughly the same as fitting on each remainder $\mathbf{T} - \mathbf{T}_k$ and correspondingly testing on fold \mathbf{T}_k , and then averaging. When N is a multiple of K , these are exactly the same.) Assuming that one has randomized the order of the cases in a training set, Figure 3 portrays the K folds and how cross-validation proceeds.

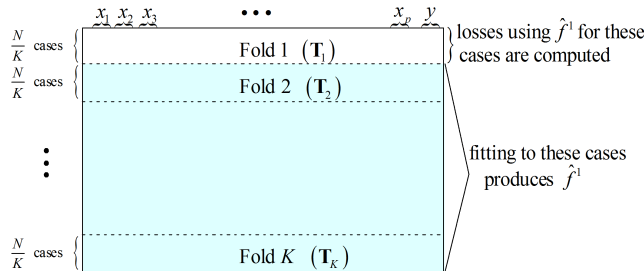


Figure 3: Schematic for K -fold cross-validation.

The choice $K = N$ is called "leave one out (LOO) cross-validation" and in this case there are sometimes slick computational ways of evaluating $CV(\hat{f})$. This has been true for ordinary least squares SEL prediction for some time and recent work of Zou and Wang has provided results for classification problems as well. As discussed more completely in Section 16.3, cross-validation actually estimates Err for a training set of size approximately $N(1 - K^{-1})$, so there is potential bias⁷ that typically decreases with increasing K , making LOO cross-validation attractive from this point of view.

⁷Note that for the use of cross-validation error to identify appropriate complexity, bias is a problem only if it is *not constant* across choices of predictors and their complexities.

Notice that unless $K = N$, even for fixed training set \mathbf{T} , $CV(\hat{f})$ is random, owing to its dependence upon random assignment of training cases to folds. It is thus highly attractive in cases where $K < N$ is used, to replace $CV(\hat{f})$ with an average cross-validation error (say $\overline{CV(\hat{f})}$) derived from a large number of repeated splittings of \mathbf{T} into K folds. The `caret` package in **R** (and, presumably, similar packages in other systems) facilitates this repeated cross-validation for a variety of prediction methods and effectively replaces $CV(\hat{f})$ with its expected value across randomizations. The fact that this averaging (and related computational burden) is not needed for LOO cross-validation is another reason to find it attractive.

LOO cross-validation has been portrayed in the statistical folklore as suffering from a large variance. The argument has been that because its \hat{f}^k are all built on nearly the same training sets and produce similar predictions, the averaging done in computing $CV(\hat{f})$ might be relatively ineffective in reducing variance. This logic has been thought to motivate bias-variance trade-off considerations for representation of Err , making $K = 5$ and $K = 10$ popular choices in practice. But the recent work of Zou and Wang has strongly called into question the truth of the folklore and makes a convincing case for LOO cross-validation when it is feasible.

1.3.7 Choosing a Predictor Based on Cross-Validation

One popular rule of thumb for choosing between predictors of differing complexities on the basis of a single K -fold cross-validation for each (with $K < N$) has been this. For the complexity producing the smallest realized cross-validation error, one computes a "standard error" for the prediction error. That is, for each fold \mathbf{T}_k , one computes a k th "test error" (call it $CV_k(\hat{f})$), for \hat{f}^k (referred to in step 2.) obtained by fitting on remainder $\mathbf{T} - \mathbf{T}_k$, evaluating on \mathbf{T}_k . Then for SD_K the sample standard deviation of $CV_1(\hat{f}), CV_2(\hat{f}), \dots, CV_K(\hat{f})$, the "standard error" of interest is SD_K/\sqrt{K} . One then selects for use the least complex predictor with its own corresponding cross-validation error no larger than

$$CV(\hat{f}) + SD_K/\sqrt{K}$$

This is sometimes called the "one standard error rule of thumb" and is presumably motivated by recognition of the uncertainty involved in cross-validation (deriving from the randomness of 1) the selection of the training set and 2) the partitioning of it into folds) and the desire to avoid overfitting. But (in light of the dependence of the $CV_k(\hat{f})$)) the validity of the supposed standard error is at best quite approximate, and then the appropriateness of a "one standard error rule" is not at all obvious.

The most obvious, aggressive, and logically defensible way of using $CV(\hat{f})$

(or $\overline{CV(\hat{f})}$) to choose a predictor is to simply use the \hat{f} minimizing the function $CV(\cdot)$ (or $\overline{CV(\cdot)}$). We will call this way of operating a "**pick-the-(cross-validation error)-winner rule.**"

It is an important and somewhat subtle point that if

$$\tilde{f} = \arg \min_{\hat{f}} CV(\hat{f})$$

the minimum cross-validation error $CV(\tilde{f})$ (or $\overline{CV(\tilde{f})}$) is **not** a valid cross-validation error for a pick-the-winner rule!⁸ The issue is that while $CV(\hat{f})$ (or $\overline{CV(\hat{f})}$) can legitimately guide the choice of \hat{f} , its use is then actually part of a larger program of "predictor development" than that represented by any single argument of $CV(\cdot)$ (or $\overline{CV(\cdot)}$). That being the case, in order to assess the likely performance of \tilde{f} , via cross-validation, **inside each remainder $T - T_k$** one must

1. **split into K folds,**
2. **fit on the K remainders,**
3. **predict on the folds and make a cross-validation error,**
4. **pick a winner for the function in 3., say \tilde{f}^k , and**
5. **then predict on T_k using \tilde{f}^k .**

It is the values $\tilde{f}^{k(i)}(\mathbf{x}_i)$ that are used in form (15) to predict the performance of a predictor derived from optimizing a cross-validation error across a set of predictors.

The basic principle at work here (and always) in making valid cross-validation errors is that **whatever one will ultimately do in the entire training set to make a predictor must be redone (in its entirety!) in every remainder and applied to the corresponding fold.**

1.3.8 Penalized Training Error Fitting and Choosing Complexity

A way of creating (and ultimately using cross-validation to choose a good value for) an explicit numerical complexity measure in supervised learning is through the notion of *penalization* of training error. That is, suppose that in the framework of Section 1.3.4 one can define for every element of the class of functions $\mathcal{S} = \{g\}$ a complexity penalty $J[g] \geq 0$ and for every $\lambda \geq 0$ defines a measure of undesirability for g reflecting both fit to the training data and complexity by

$$\overline{\text{err}} + \lambda J[g] = \frac{1}{N} \sum_{i=1}^N L(g(\mathbf{x}_i), y_i) + \lambda J[g] \quad (16)$$

⁸Intuition suggests that it will typically be optimistic as representing Err for the pick-the-winner predictor.

Call the function optimizing this objective (over choices of g) for a given λ by the name \hat{f}_λ . The smaller is λ , the more complex will be \hat{f}_λ .

As a simple example, consider $p = 1$ SEL prediction on \mathfrak{R} with standardized input x . With $\mathcal{S} = \{\beta_1 x + \beta_2 x^2 + \beta_3 x^3 | \beta_1, \beta_2, \beta_3 \text{ are all real}\}$, using $J[g] = \beta_2^2 + \beta_3^2$ penalizes lack of linearity in a fitted cubic. Small λ produces essentially least squares fitting of a cubic and large λ produces least squares fitting of a line.

Applying this penalized fitting to each remainder $\mathbf{T} - \mathbf{T}_k$ to produce K predictors \hat{f}_λ^k , one can as in display (15) derive a cross-validation error corresponding to λ as

$$CV(\lambda) = \frac{1}{N} \sum_{i=1}^N L\left(\hat{f}_\lambda^{k(i)}(\mathbf{x}_i), y_i\right)$$

To produce a pick-the-winner rule in this context, one minimizes this (or an average cross-validation error $\overline{CV(\lambda)}$ if $K < N$ is employed) by choice of λ , producing the optimizer, say, λ^{opt} (a function of the training set), and ultimately employs λ^{opt} and the criterion (16) with the whole training set (\mathbf{T}) to produce the pick-the-winner predictor $\tilde{f} = \hat{f}_{\lambda^{\text{opt}}}$ for application.

1.4 Good Features and Prediction

There are sometimes more or less standard/obvious ways for taking a small number (p) of "original" features and making (often many) additional ones. Powers of original variables x_j make sense where polynomial predictors for a quantitative y are natural. Where a quantitative y can be expected to have periodic character, sin and/or cos functions can be useful, etc.

Exactly how to think about such data preprocessing and feature-making is not always completely obvious. It is the intention here to raise several conceptual and practical issues that potentially arise in feature engineering in a supervised learning problem.

1.4.1 Classification Models and Optimal Features

Consider first a K -class classification model, where y takes values in $\mathcal{G} = \{0, 1, \dots, K-1\}$. P then has K conditional distributions for $\mathbf{x}|y$, that we will assume are specified by densities

$$p(\mathbf{x}|0), p(\mathbf{x}|1), \dots, p(\mathbf{x}|K-1)$$

(There is no loss of generality here. These could be densities with respect to the simple arithmetic average of the K class-conditional distributions.) There is important statistical theory concerning minimal sufficiency that promises that regardless of the original dimensionality of \mathbf{x} (namely, p) there is a $(K-1)$ -dimensional feature that carries all available information about y encoded in \mathbf{x} .

For $K = 2$ the 1-dimensional likelihood ratio statistic

$$\mathcal{L}(\mathbf{x}) = \frac{p(\mathbf{x}|1)}{p(\mathbf{x}|0)} \quad (17)$$

is "minimal sufficient." If one knew the value of $\mathcal{L}(\mathbf{x})$ one would know all \mathbf{x} has to say about y . An optimal single feature is $\mathcal{L}(\mathbf{x})$. In a practical problem, the closer that one can come to engineering features "like" $\mathcal{L}(\mathbf{x})$, the more efficiently/parsimoniously one represents the input vector \mathbf{x} . Of course, any monotone transform of $\mathcal{L}(\mathbf{x})$ is equally as good as $\mathcal{L}(\mathbf{x})$.

For $K > 2$, roughly speaking the $K-1$ ratios $p(\mathbf{x}|k)/p(\mathbf{x}|0)$ (taken together) form a minimal sufficient statistic for the model. This potentially isn't quite true because of possible problems where $p(\mathbf{x}|0) = 0$. But it is true that with $s(\mathbf{x}) = \sum_{k=0}^{K-1} p(\mathbf{x}|k)$ the vector

$$\left(\frac{p(\mathbf{x}|1)}{s(\mathbf{x})}, \frac{p(\mathbf{x}|2)}{s(\mathbf{x})}, \dots, \frac{p(\mathbf{x}|K-1)}{s(\mathbf{x})} \right) \quad (18)$$

(and many variants of it) is (are) minimal sufficient. To the extent that one can engineer features approximating these $K-1$ ratios⁹, one can parsimoniously represent the input vector.

1.4.2 Approximating "Partially Optimal" Numerical Features for Discrete Parts of Input Vectors

When one or more coordinates x_j of an input vector \mathbf{x} are categorical, ordinal, or numerical-but-discrete it can be useful to try to represent the information they together provide about y in terms of a (low-dimensional) feature taking values in \mathfrak{R}^q for a relatively small q . Numerical features are simply more directly handled by standard prediction methodologies than categorical, ordinal, or even discrete numerical ones. Here we consider low-dimensional "partially optimal" numerical features based on vectors of categorical, ordinal, and/or discrete numerical inputs and empirical approximations to them.

Suppose that a sub-vector of \mathbf{x} , say $\tilde{\mathbf{x}} = (x_{j_1}, x_{j_2}, \dots, x_{j_D})$, has entries with respectively only finite numbers M_1, M_2, \dots, M_D of possible values, so that the sub-vector has $M = M_1 \cdot M_2 \cdot \dots \cdot M_D$ possible values. One standard way of representing such an $\tilde{\mathbf{x}}$ is through the use of $M-1$ dummy (0-1) variables, one for every possible value of $\tilde{\mathbf{x}}$ except an arbitrarily chosen "last" one. That deals with the possibility that parts of $\tilde{\mathbf{x}}$ are ordinal or categorical with more than 2 possible values in terms of making arithmetic operations applied to their representations sensible. But it also explodes the number of features representing $\tilde{\mathbf{x}}$ from D to M , motivating contemplation of another approach.

Consider then the case of classification models where y takes values in a finite set $\mathcal{G} = \{0, 1, \dots, K-1\}$. There are $M \cdot K$ possible values of $(\tilde{\mathbf{x}}, y)$. Then (based on all or a fixed subset of the full training set) with $N_{\tilde{\mathbf{x}}, y}$ the number of

⁹These are the $K-1$ conditional probabilities $P[y=1|\mathbf{x}], \dots, P[y=K-1|\mathbf{x}]$ for the case where each $P[y=k] = 1/K$.

training cases with $\tilde{\mathbf{x}}_i = \tilde{\mathbf{x}}$ and $y_i = y$ (\cdot), let $N_{\cdot,y} = \sum_{\tilde{\mathbf{x}}} N_{\tilde{\mathbf{x}},y}$ be the number of training cases with $\tilde{\mathbf{x}}_i = \tilde{\mathbf{x}}$ and $N_{\tilde{\mathbf{x}},\cdot} = \sum_y N_{\tilde{\mathbf{x}},y}$ be the number of training cases with $y_i = y$. The vector function of $\tilde{\mathbf{x}}$

$$\hat{\mathbf{P}}(y|\tilde{\mathbf{x}}) = \frac{1}{N_{\tilde{\mathbf{x}},\cdot}} (N_{\tilde{\mathbf{x}},1}, N_{\tilde{\mathbf{x}},2}, \dots, N_{\tilde{\mathbf{x}},K-1}) \quad (19)$$

serves as an approximation to the numerical $(K-1)$ -dimensional feature with entries $P[y = k|\tilde{\mathbf{x}}]$. And with $\hat{s}(\tilde{\mathbf{x}}) = \sum_{k=0}^{K-1} (N_{\tilde{\mathbf{x}},k}/N_{\cdot,k})$, the vector function of $\tilde{\mathbf{x}}$

$$\hat{\mathbf{L}}(\tilde{\mathbf{x}}) = \frac{1}{s(\tilde{\mathbf{x}})} \left(\frac{N_{\tilde{\mathbf{x}},1}}{N_{\cdot,1}}, \frac{N_{\tilde{\mathbf{x}},2}}{N_{\cdot,2}}, \dots, \frac{N_{\tilde{\mathbf{x}},K-1}}{N_{\cdot,K-1}} \right) \quad (20)$$

serves as an approximation to the numerical $(K-1)$ -dimensional feature with entries $p(\tilde{\mathbf{x}}|k) / \sum_{k=0}^{K-1} p(\tilde{\mathbf{x}}|k)$.

We noted in the previous section that the vector function with entries $p(\mathbf{x}|k) / \sum_{k=0}^{K-1} p(\mathbf{x}|k)$ is minimal sufficient in the classification model. And the vector with entries $P[y = k|\mathbf{x}]$ is an optimal predictor under cross-entropy loss and a function of it is an optimal 0-1 loss classifier. The versions of these in displays (19) and (20) based on $\tilde{\mathbf{x}}$ (rather than the full input vector \mathbf{x}) thus might then be considered "partially optimal," representing the best one could do supplied only with the discrete part, $\tilde{\mathbf{x}}$, of \mathbf{x} . And then $\hat{\mathbf{P}}(y|\tilde{\mathbf{x}})$ and/or $\hat{\mathbf{L}}(\tilde{\mathbf{x}})$ are *approximate* partially optimal numerical features of low dimension. How useful they will be in practice will depend in part upon how large are the values $N_{\tilde{\mathbf{x}},y}$, which in turn depends upon how large the (whole or partial) training set is in comparison to M . As values of D and M employed increase, one should expect the effectiveness of the "partially optimal" features to increase and the fidelity of the approximations to them to decrease. Some trade-off between these effects will be necessary and a sensible way to try and employ this idea in practice is to build sets of these features with a spectrum of values M and look for one that is overall most effective.

Now drop the assumption that y has only K values and consider what in this direction can be done in SEL prediction problems. We have noted repeatedly that here the theoretically optimal predictor of y is $f(\mathbf{x}) = E[y|\mathbf{x}]$, in some sense an "unrealizable optimal feature" for prediction. By the same token, if one had access to only $\tilde{\mathbf{x}}$, an optimal feature for prediction would be $E[y|\tilde{\mathbf{x}}]$. That suggests thinking of this conditional mean function as a "partially optimal" 1-dimensional feature for encoding the information in $\tilde{\mathbf{x}}$ in the full prediction problem.

Simple approximation to the function $E[y|\tilde{\mathbf{x}}]$ based on (all or part of) a training set is straightforward and can be an effective way to make a 1-dimensional numerical feature to represent the D -dimensional $\tilde{\mathbf{x}}$. With $N_{\tilde{\mathbf{x}}}$ the number of training cases with $\tilde{\mathbf{x}}_i = \tilde{\mathbf{x}}$ (based on all or a subset of the full training set), the

corresponding empirical mean output

$$\bar{y}(\tilde{\mathbf{x}}) = \frac{1}{N_{\tilde{\mathbf{x}}}} \sum_{i \text{ with } \tilde{\mathbf{x}}_i = \tilde{\mathbf{x}}} y_i$$

across the M possible values of $\tilde{\mathbf{x}}$ defines an *approximate* partially optimal feature for SEL prediction. How useful this will be in practice will depend in part upon how large the values $N_{\tilde{\mathbf{x}}}$ are, which in turn depends upon how large the (whole or partial) training set is in comparison to M . As values of D and M employed increase, one should expect the effectiveness of the feature $E[y|\tilde{\mathbf{x}}]$ to increase and the fidelity of $\bar{y}(\tilde{\mathbf{x}})$ as an approximation to it to decrease. Again, some trade-off between these effects seems necessary, and building and comparing the performance of sets of these features with a spectrum of values M seems sensible.

We have here repeatedly used phrases like "all or part of a training set." It is not clear when it will be best to use an entire training set to make these empirical approximations $\hat{P}(y|\tilde{\mathbf{x}})$, $\hat{L}(\tilde{\mathbf{x}})$, or $\bar{y}(\tilde{\mathbf{x}})$, and when it will be best to reserve only a part of the training set to make them and then use the balance for predictor-building based on these approximately partially optimal features (and other features as appropriate). ISU use of a variant of the approximations $\hat{L}(\tilde{\mathbf{x}})$ built on a part of a training set reserved exclusively for feature engineering led to an international first place in the 2014 Prudsys AG Data mining Cup. The team's intuition was that both using a training case in making the function $\hat{L}(\tilde{\mathbf{x}})$ and then subsequently using the case for fitting classifier was likely to cause overfit and poor performance on test cases.

Ultimately, questions of what fraction (if any) of a training set to reserve for feature-making, which discrete sub-vector or sub-vectors to use in the development of approximate partially optimal features, and all questions of subsequent predictor fitting should in practice be answered via cross-validation (that does every detail of making predictions on K remainders and tests on the corresponding K folds). Cross-validation of *all*—including potential data splitting, choice of M , and feature making—is needed to empirically gauge likely performance on new cases. See remarks at the end of Section 1.4.5 for a bit more on this issue.

1.4.3 Abstract Feature Spaces (of Functions) and "Kernels"

There are surely situations where what P encodes about a relationship between \mathbf{x} and y is very complicated and "non-linear" (whatever that might mean in this context). Standard (and really, almost all tractable) mathematics of prediction often relies on "linear" operations: additions of vectors, multiplication of vectors by scalars, inner products (and associated norms and distances), etc. "Ordinary" creation of features can be thought of as a way to map a feature space \mathfrak{R}^p (non-linearly) to a higher-dimensional (Euclidean and therefore linear) feature space \mathfrak{R}^q . But sometimes that is ineffective because q large enough to in theory allow for good prediction based on linear operations is so large as to make an appropriate transform from \mathfrak{R}^p to \mathfrak{R}^q impossible to identify and/or use.

A very clever and practically powerful development in machine learning has been the realization that for some purposes, it is not necessary to map from \mathfrak{R}^p to a Euclidean space, but that mapping to a linear space **of functions** may be helpful. That is, creation of new numerical features based on input vector \mathbf{x} can be thought of as transformation

$$T : \mathfrak{R}^p \rightarrow \mathfrak{R}^q$$

where relationships in \mathfrak{R}^q or predictors mapping from \mathfrak{R}^q and producing a \hat{y} are then thought of as defining ones for \mathbf{x} s in \mathfrak{R}^p by simply applying T to \mathbf{x} s of interest. This line of reasoning doesn't depend at all upon T mapping to a Euclidean space. If \mathcal{A} is an abstract feature space of functions (that is an inner product space¹⁰) one might think of mapping

$$T : \mathfrak{R}^p \rightarrow \mathcal{A}$$

and using linear operations and relationships in \mathcal{A} to make relationships and predictors based on \mathbf{a} s in \mathcal{A} , and then defining corresponding ones for \mathbf{x} s in \mathfrak{R}^p by simply applying T to \mathbf{x} s of interest. After all, in some sense functions are really just high-dimensional vectors, and if transforming $\mathfrak{R}^p \rightarrow \mathfrak{R}^q$ with $p < q$ is often useful, so also might be transforming $\mathfrak{R}^p \rightarrow \mathcal{A}$.

This line of argument has especially been taken advantage of through the use of so-called "kernel functions." (Be careful. There are many different usages of the word "kernel" in the machine learning world.) Suppose that a symmetric function $\mathcal{K}(\mathbf{x}, \mathbf{z})$ with domain some part of $\mathfrak{R}^p \times \mathfrak{R}^p$ is non-negative definite in the sense that for any training set \mathbf{T} the (symmetric) $N \times N$ so-called "Gram matrix"

$$\mathbf{K} = (\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j))_{\substack{i=1, \dots, N \\ j=1, \dots, N}} \quad (21)$$

is non-negative definite. Then the space of functions that are linear combinations of "slices" of $\mathcal{K}(\mathbf{x}, \mathbf{z})$, i.e. functions of \mathbf{x} of the form

$$\sum_{j=1}^M c_j \mathcal{K}(\mathbf{x}, \mathbf{z}_j)$$

for $M > 0$ real numbers c_1, c_2, \dots, c_M , and elements $\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_M$ of \mathfrak{R}^p form a linear space (call it \mathcal{A}). It is possible to coherently define a very convenient inner product on that space starting from the basic relationship

$$\langle \mathcal{K}(\cdot, \mathbf{z}_1), \mathcal{K}(\cdot, \mathbf{z}_2) \rangle_{\mathcal{A}} \equiv \mathcal{K}(\mathbf{z}_1, \mathbf{z}_2) \quad (22)$$

and using the bilinearity of any inner product to see that then of necessity

$$\left\langle \sum_{j=1}^M c_{1j} \mathcal{K}(\cdot, \mathbf{z}_j), \sum_{j=1}^M c_{2j} \mathcal{K}(\cdot, \mathbf{z}_j) \right\rangle_{\mathcal{A}} = \sum_{j=1}^{M_1} \sum_{j'=1}^{M_2} c_{1j} c_{2j'} \mathcal{K}(\mathbf{z}_j, \mathbf{z}_{j'}) = \mathbf{c}'_1 \mathbf{K} \mathbf{c}_2$$

¹⁰See Section 2.1 for more concerning the meaning of this language.

for $\mathbf{c}'_1 = (c_{11}, \dots, c_{1M})$, $\mathbf{c}'_2 = (c_{21}, \dots, c_{2M})$, and $M \times M$ matrix \mathbf{K} with entries $\mathcal{K}(\mathbf{z}_i, \mathbf{z}_j)$. This has the important special case that for $\mathbf{c} = \mathbf{c}_1 = \mathbf{c}_2$

$$\left\| \sum_{j=1}^M c_j \mathcal{K}(\cdot, \mathbf{z}_j) \right\|_{\mathcal{A}}^2 = \left\langle \sum_{j=1}^M c_j \mathcal{K}(\cdot, \mathbf{z}_j), \sum_{j=1}^M c_j \mathcal{K}(\cdot, \mathbf{z}_j) \right\rangle_{\mathcal{A}} = \mathbf{c}' \mathbf{K} \mathbf{c}$$

Of course, since \mathcal{K} defines the inner product in \mathcal{A} it also defines the distance between $\sum_{j=1}^M c_{1j} \mathcal{K}(\cdot, \mathbf{z}_j)$ and $\sum_{j=1}^M c_{2j} \mathcal{K}(\cdot, \mathbf{z}_j)$

$$d_{\mathcal{A}} \left(\sum_{j=1}^M c_{1j} \mathcal{K}(\cdot, \mathbf{z}_j), \sum_{j=1}^M c_{2j} \mathcal{K}(\cdot, \mathbf{z}_j) \right) = \sqrt{(\mathbf{c}_1 - \mathbf{c}_2)' \mathbf{K} (\mathbf{c}_1 - \mathbf{c}_2)}$$

(with \mathbf{c}_1 , \mathbf{c}_2 , and \mathbf{K} as before).

Relationship (22) is the origin of the language that \mathcal{K} serves as a **reproducing kernel**. It both defines the linear space of functions of interest and provides the inner product for the space. Under some conditions, the space \mathcal{A} (whose elements are functions $\mathfrak{R}^p \rightarrow \mathfrak{R}$) can be extended to include *limits* of finite linear combinations of slices of the kernel function $\mathcal{K}(\cdot, \cdot)$ and the resulting construct is termed a Reproducing Kernel (Hilbert) Space (**RKHS**) of functions.

In any event, having identified an inner product space associated with a kernel, the abstract transform $T : \mathfrak{R}^p \rightarrow \mathcal{A}$ is defined by

$$T(\mathbf{x})(\cdot) = \mathcal{K}(\mathbf{x}, \cdot)$$

(remember here that $T(\mathbf{x})(\cdot)$ is a function of " \cdot "). The inner product in \mathcal{A} of two images of elements of \mathfrak{R}^p is

$$\langle T(\mathbf{x}), T(\mathbf{z}) \rangle_{\mathcal{A}} = \mathcal{K}(\mathbf{x}, \mathbf{z})$$

and for a training set with inputs $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ the span of $\{T(\mathbf{x}_i)\}_{i=1, \dots, N}$ is a linear subspace of \mathcal{A} .

Probably the most used kernel function in machine learning is the "Gaussian kernel"

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = \exp\left(-\gamma \|\mathbf{x} - \mathbf{z}\|^2\right)$$

that produces abstract features

$$T(\mathbf{x})(\cdot) = \exp\left(-\gamma \|\mathbf{x} - \cdot\|^2\right)$$

that are radially symmetric p -variate Normal density functions located at \mathbf{x} . The function space consists of linear combinations of such functions (and limits of them) and the abstract inner product of $T(\mathbf{x})$ and $T(\mathbf{z})$ is $\exp\left(-\gamma \|\mathbf{x} - \mathbf{z}\|^2\right)$.

One can even give up requiring that the domain of a kernel function $\mathcal{K}(\mathbf{x}, \mathbf{z})$ is a subset of $\mathfrak{R}^p \times \mathfrak{R}^p$, replacing it with arbitrary $\mathcal{X} \times \mathcal{X}$ and requiring *only* that the Gram matrix be non-negative definite for any set of $\{\mathbf{x}_i\}_{i=1}^n$, $\mathbf{x}_i \in \mathcal{X}$. It is in this context that the "string kernels" of "text processing" briefly discussed in Section 1.4.4 can be called "kernels."

Kernel Mechanics A direct way of producing a kernel function is through a Euclidean inner product of vectors of "features." That is, if $\phi : \mathcal{X} \rightarrow \mathbb{R}^m$ (so that component j of ϕ , ϕ_j , creates the univariate real feature $\phi_j(\mathbf{x})$) then for $\langle \cdot, \cdot \rangle$ the usual Euclidean inner product (dot product),

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle \quad (23)$$

is a kernel function. (This basic idea will be used in Section 2.4.2.)

Section 6.2 of the book *Pattern Recognition and Machine Learning* by Bishop notes that it is very easy to make new kernel functions from known ones. In particular, for $c > 0$, $\mathcal{K}_1(\cdot, \cdot)$ and $\mathcal{K}_2(\cdot, \cdot)$ kernel functions on $\mathcal{X} \times \mathcal{X}$, $h(\cdot) : \mathcal{X} \rightarrow \mathbb{R}$ arbitrary, $q(\cdot)$ a polynomial with non-negative coefficients, $\phi : \mathcal{X} \rightarrow \mathbb{R}^m$, $\mathcal{K}_3(\cdot, \cdot)$ a kernel on $\mathbb{R}^m \times \mathbb{R}^m$, and \mathbf{M} a non-negative definite matrix, all of the following are kernel functions:

1. $\mathcal{K}(\mathbf{x}, \mathbf{z}) = c\mathcal{K}_1(\mathbf{x}, \mathbf{z})$ on $\mathcal{X} \times \mathcal{X}$,
2. $\mathcal{K}(\mathbf{x}, \mathbf{z}) = h(\mathbf{x})\mathcal{K}_1(\mathbf{x}, \mathbf{z})h(\mathbf{z})$ on $\mathcal{X} \times \mathcal{X}$,
3. $\mathcal{K}(\mathbf{x}, \mathbf{z}) = q(\mathcal{K}_1(\mathbf{x}, \mathbf{z}))$ on $\mathcal{X} \times \mathcal{X}$,
4. $\mathcal{K}(\mathbf{x}, \mathbf{z}) = \exp(\mathcal{K}_1(\mathbf{x}, \mathbf{z}))$ on $\mathcal{X} \times \mathcal{X}$,
5. $\mathcal{K}(\mathbf{x}, \mathbf{z}) = \mathcal{K}_1(\mathbf{x}, \mathbf{z}) + \mathcal{K}_2(\mathbf{x}, \mathbf{z})$ on $\mathcal{X} \times \mathcal{X}$,
6. $\mathcal{K}(\mathbf{x}, \mathbf{z}) = \mathcal{K}_1(\mathbf{x}, \mathbf{z})\mathcal{K}_2(\mathbf{x}, \mathbf{z})$ on $\mathcal{X} \times \mathcal{X}$,
7. $\mathcal{K}(\mathbf{x}, \mathbf{z}) = \mathcal{K}_3(\phi(\mathbf{x}), \phi(\mathbf{z}))$ on $\mathcal{X} \times \mathcal{X}$, and
8. $\mathcal{K}(\mathbf{x}, \mathbf{z}) = \mathbf{x}'\mathbf{M}\mathbf{z}$ on $\mathbb{R}^m \times \mathbb{R}^m$.

(Fact 7 generalizes the basic insight of display (23).) Further, if $\mathcal{X} \subset \mathcal{X}_A \times \mathcal{X}_B$ and $\mathcal{K}_A(\cdot, \cdot)$ is a kernel on $\mathcal{X}_A \times \mathcal{X}_A$ and $\mathcal{K}_B(\cdot, \cdot)$ is a kernel on $\mathcal{X}_B \times \mathcal{X}_B$, then the following are both kernel functions:

9. $\mathcal{K}((\mathbf{x}_A, \mathbf{x}_B), (\mathbf{z}_A, \mathbf{z}_B)) = \mathcal{K}_A(\mathbf{x}_A, \mathbf{z}_A) + \mathcal{K}_B(\mathbf{x}_B, \mathbf{z}_B)$ on $\mathcal{X} \times \mathcal{X}$, and
10. $\mathcal{K}((\mathbf{x}_A, \mathbf{x}_B), (\mathbf{z}_A, \mathbf{z}_B)) = \mathcal{K}_A(\mathbf{x}_A, \mathbf{z}_A)\mathcal{K}_B(\mathbf{x}_B, \mathbf{z}_B)$ on $\mathcal{X} \times \mathcal{X}$.

An example of a kernel on a somewhat abstract (but finite) space is this. For a finite set \mathcal{B} consider $\mathcal{X} = 2^{\mathcal{B}}$, the set of all subsets of \mathcal{B} . A kernel on $\mathcal{X} \times \mathcal{X}$ can then be defined by

$$\mathcal{K}(B_1, B_2) = 2^{|B_1 \cap B_2|} \quad \text{for } B_1 \subset \mathcal{B} \text{ and } B_2 \subset \mathcal{B}$$

There are several probabilistic and statistical arguments that can lead to forms for kernel functions. For example, a useful fact from probability theory (Bochner's Theorem) says that characteristic functions for p -dimensional distributions are non-negative definite complex-valued functions of $\mathbf{s} \in \mathbb{R}^p$. So if $\psi(\mathbf{s})$ is a *real-valued* characteristic function, then

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = \psi(\mathbf{x} - \mathbf{z})$$

is a kernel function on $\mathfrak{R}^p \times \mathfrak{R}^p$. Related to this line of thinking are lists of standard characteristic functions (that in turn produce kernel functions) and theorems about conditions sufficient to guarantee that a real-valued function is a characteristic function. For example, each of the following is a real characteristic function for a *univariate* random variable (that can lead to a kernel on $\mathfrak{R}^1 \times \mathfrak{R}^1$):

1. $\psi(t) = \cos at$ for some $a > 0$,
2. $\psi(t) = \frac{\sin at}{at}$ for some $a > 0$,
3. $\psi(t) = \exp(-at^2)$ for some $a > 0$, and
4. $\psi(t) = \exp(-a|t|)$ for some $a > 0$.

And one theorem about sufficient conditions for a real-valued function on \mathfrak{R}^1 to be a characteristic function says that if ψ is symmetric ($\psi(-t) = \psi(t)$), $\psi(0) = 1$, and ψ is decreasing and convex on $[0, \infty)$, then ψ is the characteristic function of some distribution on \mathfrak{R}^1 . (See Chung's *A Course in Probability Theory*, page 191.)

Bishop points out two constructions motivated by statistical modeling that yield kernels that have been used in the machine learning literature. One is this. For a parametric model on (a potentially completely abstract) \mathcal{X} , consider densities $p(\mathbf{x}|\boldsymbol{\theta})$ that when treated as functions of $\boldsymbol{\theta}$ are likelihood functions (for various possible observed \mathbf{x}). Then for a distribution G for $\boldsymbol{\theta} \in \Theta$,

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = \int p(\mathbf{x}|\boldsymbol{\theta}) p(\mathbf{z}|\boldsymbol{\theta}) dG(\boldsymbol{\theta})$$

is a kernel. This is the inner product in the space of square integrable functions on the probability space Θ with measure G of the two likelihood functions. In this space, the distance between the functions (of $\boldsymbol{\theta}$) $p(\mathbf{x}|\boldsymbol{\theta})$ and $p(\mathbf{z}|\boldsymbol{\theta})$ is

$$\sqrt{\int (p(\mathbf{x}|\boldsymbol{\theta}) - p(\mathbf{z}|\boldsymbol{\theta}))^2 dG(\boldsymbol{\theta})}$$

and what is going on here is the implicit use of (infinite-dimensional) features that are likelihood functions for the "observations" \mathbf{x} . Once one starts down this path, other possibilities come to mind. One is to replace likelihoods with loglikelihoods and consider the issue of "centering" and even "standardization." That is, one might define a feature (a function of $\boldsymbol{\theta}$) corresponding to \mathbf{x} as

$$\phi_{\mathbf{x}}(\boldsymbol{\theta}) = \ln p(\mathbf{x}|\boldsymbol{\theta}) \quad \text{or} \quad \phi'_{\mathbf{x}}(\boldsymbol{\theta}) = \ln p(\mathbf{x}|\boldsymbol{\theta}) - \int \ln p(\mathbf{x}|\boldsymbol{\theta}) dG(\boldsymbol{\theta})$$

or even
$$\phi''_{\mathbf{x}}(\boldsymbol{\theta}) = \frac{\ln p(\mathbf{x}|\boldsymbol{\theta}) - \int \ln p(\mathbf{x}|\boldsymbol{\theta}) dG(\boldsymbol{\theta})}{\sqrt{\int (\ln p(\mathbf{x}|\boldsymbol{\theta}) - \int \ln p(\mathbf{x}|\boldsymbol{\theta}) dG(\boldsymbol{\theta}))^2 dG(\boldsymbol{\theta})}}$$

Then obviously, the corresponding kernel function is

$$\begin{aligned} \mathcal{K}(\mathbf{x}, \mathbf{z}) &= \int \phi_{\mathbf{x}}(\boldsymbol{\theta}) \phi_{\mathbf{z}}(\boldsymbol{\theta}) dG(\boldsymbol{\theta}) \quad \text{or} \quad \mathcal{K}'(\mathbf{x}, \mathbf{z}) = \int \phi'_{\mathbf{x}}(\boldsymbol{\theta}) \phi'_{\mathbf{z}}(\boldsymbol{\theta}) dG(\boldsymbol{\theta}) \\ \text{or} \quad \mathcal{K}''(\mathbf{x}, \mathbf{z}) &= \int \phi''_{\mathbf{x}}(\boldsymbol{\theta}) \phi''_{\mathbf{z}}(\boldsymbol{\theta}) dG(\boldsymbol{\theta}) \end{aligned}$$

(Of these three possibilities, centering alone is probably the most natural from a statistical point of view. It is the "shape" of a loglikelihood that is important in statistical context, not its absolute level. Two loglikelihoods that differ by a constant are equivalent for most statistical purposes. Centering perfectly lines up two loglikelihoods that differ by a constant.)

In a regular statistical model for \mathbf{x} taking values in \mathcal{X} with Euclidean parameter vector $\boldsymbol{\theta} = (\theta_1, \theta_2, \dots, \theta_k)$, the $k \times k$ Fisher information matrix, say $I(\boldsymbol{\theta})$, is non-negative definite. Then with score function

$$\nabla_{\boldsymbol{\theta}} \ln p(\mathbf{x}|\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_1} \ln p(\mathbf{x}|\boldsymbol{\theta}) \\ \frac{\partial}{\partial \theta_2} \ln p(\mathbf{x}|\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_k} \ln p(\mathbf{x}|\boldsymbol{\theta}) \end{pmatrix}$$

(for any fixed $\boldsymbol{\theta}$) the function

$$\mathcal{K}_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) = \nabla_{\boldsymbol{\theta}} \ln p(\mathbf{x}|\boldsymbol{\theta})' (I(\boldsymbol{\theta}))^{-1} \nabla_{\boldsymbol{\theta}} \ln p(\mathbf{z}|\boldsymbol{\theta})$$

has been called the "Fisher kernel" in the machine learning literature. (It follows from Bishops's 7. and 8. that this is indeed a kernel function.) Note that $\mathcal{K}_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x})$ is essentially the score test statistic for a point null hypothesis about $\boldsymbol{\theta}$. The implicit feature vector here is the k -dimensional score function (evaluated at some fixed $\boldsymbol{\theta}$, a basis for testing about $\boldsymbol{\theta}$), and rather than Euclidean norm, the norm $\|\mathbf{u}\|_{\boldsymbol{\theta}} \equiv \sqrt{\mathbf{u}'(I(\boldsymbol{\theta}))^{-1} \mathbf{u}}$ is implicitly in force for judging the size of differences in feature vectors.

1.4.4 Document Features and String Kernels for Text Processing

An important application of various kinds of both supervised and unsupervised learning methods is that of text processing. The object is to quantify structure and commonalities in text documents. Patterns in characters and character strings and words are used to characterize documents, group them into clusters, and classify them into types. We here say a bit about some simple methods that have been applied.

Suppose that N documents in a collection (or corpus) are under study. One needs to define "features" for these, or at least some kind of "kernel" functions for computing the inner products required for producing principal components

in an implicit feature space (and subsequently clustering or deriving classifiers, and so on).

If one treats documents as simply sets of words (ignoring spaces and punctuation and any kind of order of words) one simple set of features for documents $\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_N$ is a set of counts of word frequencies. That is, for a set of p words appearing in at least one document, one might take

$$x_{ij} = \text{the number of occurrences of word } j \text{ in document } i$$

and operate on a representation of the documents in terms of an $N \times p$ data matrix \mathbf{X} . These raw counts x_{ij} are often transformed before processing. One popular idea is the use of a "tf-idf" (term frequency-inverse document frequency) weighting of elements of \mathbf{X} . This replaces x_{ij} with

$$t_{ij} = x_{ij} \ln \frac{N}{\sum_{i=1}^N I[x_{ij} > 0]}$$

or variants thereof. (This up-weights non-zero counts of words that occur in few documents. The logarithm is there to prevent this up-weighting from overwhelming all other aspects of the counts.) One might also decide that document length is a feature that is not really of primary interest and determine to normalize vectors \mathbf{x}_i (or \mathbf{t}_i) in one way or another. That is, one might begin with values

$$\frac{x_{ij}}{\sum_{j=1}^p x_{ij}} \quad \text{or} \quad \frac{x_{ij}}{\|\mathbf{x}_i\|}$$

rather than values x_{ij} . This latter consideration is, of course, not relevant if the documents in the corpus all have roughly the same length.

Processing methods that are based only on variants of the word counts x_{ij} are usually said to be based on the "Bag-of-Words." They obviously ignore potentially important word order. (The instructions "turn right then left" and "turn left then right" are obviously quite different instructions.) One could then consider ordered pairs or n -tuples of words.

So, with some "alphabet" \mathbb{A} (that might consist of English words, Roman letters, amino acids in protein sequencing, base pairs in DNA sequencing, etc.) consider strings of elements of elements of the alphabet, say

$$\mathbf{s} = b_1 b_2 \cdots b_{|\mathbf{s}|} \quad \text{where each } b_i \in \mathbb{A}$$

A document (... or protein sequence ... or DNA sequence) might be idealized as such a string of elements of \mathbb{A} . An n -gram in this context is simply a string of n elements of \mathbb{A} , say

$$\mathbf{u} = b_1 b_2 \cdots b_n \quad \text{where each } b_i \in \mathbb{A}$$

Frequencies of unigrams (1-grams) in documents are (depending upon the alphabet) "bag-of-words" statistics for words or letters or amino acids, etc. Use of features of documents that are counts of occurrences of all possible n -grams appears to often be problematic, because unless $|\mathbb{A}|$ and n are fairly small, $p = |\mathbb{A}|^n$

will be huge and then \mathbf{X} huge and sparse (for ordinary N and $|\mathbf{s}|$). And in many contexts, sequence/order structure is not so "local" as to be effectively expressed by only frequencies of n -grams for small n .

One idea that seems to be currently popular is to define a set of interesting strings, say $\mathcal{U} = \{\mathbf{u}_i\}_{i=1}^p$ and look for their occurrence anywhere in a document, with the understanding that they may be realized as substrings of longer strings. That is, when looking for string \mathbf{u} (of length n) in a document \mathbf{s} , we count every different substring of \mathbf{s} (say $\mathbf{s}' = s_{i_1}s_{i_2}\cdots s_{i_n}$) for which

$$\mathbf{s}' = \mathbf{u}$$

But we discount those substrings of \mathbf{s} matching \mathbf{u} according to length as follows. For some $\lambda > 0$ (the choice $\lambda = .5$ seems pretty common) give matching substring $\mathbf{s}' = s_{i_1}s_{i_2}\cdots s_{i_n}$ weight

$$\lambda^{i_n - i_1 + 1} = \lambda^{|\mathbf{s}'|}$$

so that document i (represented by string \mathbf{s}_i) gets value of feature j

$$x_{ij} = \sum_{s_{i_1}s_{i_2}\cdots s_{i_n}=\mathbf{u}_j} \lambda^{l_n - l_1 + 1} \quad (24)$$

It further seems that it's common to normalize the rows of \mathbf{X} by the usual Euclidean norm, producing in place of x_{ij} the value

$$\frac{x_{ij}}{\|\mathbf{x}_i\|} \quad (25)$$

This notion of using features (24) or normalized features (25) looks attractive, but potentially computationally prohibitive, particularly since the "interesting set" of strings \mathcal{U} is often taken to be \mathbb{A}^n . One doesn't want to have to compute all features (24) directly and then operate with the very large matrix \mathbf{X} . But just as we were reminded in Section 2.4.2, it is only $\mathbf{X}\mathbf{X}'$ that is required to find principal components of the features (or to define SVM classifiers or any other classifiers or clustering algorithms based on principal components). So if there is a way to efficiently compute or approximate inner products for rows of \mathbf{X} defined by form (24), namely

$$\begin{aligned} \langle \mathbf{x}_i, \mathbf{x}_{i'} \rangle &= \sum_{\mathbf{u} \in \mathbb{A}^n} \left(\sum_{s_{i_1}s_{i_2}\cdots s_{i_n}=\mathbf{u}} \lambda^{l_n - l_1 + 1} \right) \left(\sum_{s_{i'_1}s_{i'_2}\cdots s_{i'_n}=\mathbf{u}} \lambda^{m_n - m_1 + 1} \right) \\ &= \sum_{\mathbf{u} \in \mathbb{A}^n} \sum_{s_{i_1}s_{i_2}\cdots s_{i_n}=\mathbf{u}} \sum_{s_{i'_1}s_{i'_2}\cdots s_{i'_n}=\mathbf{u}} \lambda^{l_n - l_1 + m_n - m_1 + 2} \end{aligned}$$

it might be possible to employ this idea. And if the inner products $\langle \mathbf{x}_i, \mathbf{x}_{i'} \rangle$ can be computed efficiently, then so can the inner products

$$\left\langle \frac{1}{\|\mathbf{x}_i\|} \mathbf{x}_i, \frac{1}{\|\mathbf{x}_{i'}\|} \mathbf{x}_{i'} \right\rangle = \frac{\langle \mathbf{x}_i, \mathbf{x}_{i'} \rangle}{\sqrt{\langle \mathbf{x}_i, \mathbf{x}_i \rangle \langle \mathbf{x}_{i'}, \mathbf{x}_{i'} \rangle}}$$

needed to employ $\mathbf{X}\mathbf{X}'$ for the normalized features (25). For what it is worth, it is in vogue to call the function of documents \mathbf{s} and \mathbf{t} defined by

$$\mathcal{K}(\mathbf{s}, \mathbf{t}) = \sum_{\mathbf{u} \in \mathbb{A}^n} \sum_{s_{l_1} s_{l_2} \dots s_{l_n} = \mathbf{u}} \sum_{t_{m_1} t_{m_2} \dots t_{m_n} = \mathbf{u}} \lambda^{l_n - l_1 + m_n - m_1 + 2}$$

the String Subsequence Kernel and then call the matrix $\mathbf{X}\mathbf{X}' = (\langle \mathbf{x}_i, \mathbf{x}_j \rangle) = (\mathcal{K}(\mathbf{s}_i, \mathbf{s}_j))$ the Gram matrix for that "kernel." The good news is that there are fairly simple recursive methods for computing $\mathcal{K}(\mathbf{s}, \mathbf{t})$ exactly in $O(n|\mathbf{s}||\mathbf{t}|)$ time and that there are approximations that are even faster (see the 2002 *Journal of Machine Learning Research* paper of Lodhi *et al.*). That makes the implicit use of features (24) or normalized features (25) possible in many text processing problems.

1.4.5 "Feature Engineering" and Data "Pre-processing": More Perspective and Prediction of Predictor Efficacy

Feature engineering amounts to replacing every observation vector \mathbf{x} with a fixed function/transform thereof, $T(\mathbf{x})$. It should then be completely obvious that feature engineering cannot produce training data that are intrinsically "more informative" than the original ones. In fact, if the function $T(\cdot)$ is not one-to-one, a transformed dataset is potentially *less* informative than the original in the absolute sense of its potential usefulness.¹¹ What then is the point of feature engineering? It is to put data into a form compatible with simple existing methods of processing inputs into outputs or to provide additional predictors beyond what standard methods produce when applied directly. It is a common form of sloppy thought or expression to say that feature engineering makes data more informative. Rather, it can make them more compatible with standard prediction methodologies than the original training set¹² or extend the flexibility of those standard prediction methodologies.

As a toy example, consider a 2-class classification problem with $\mathbf{x} \in \mathfrak{R}^2$ where every training case with $\|\mathbf{x}_i - (2, 2)'\| < 1$ has $y_i = 0$ and every training case with $\|\mathbf{x}_i - (2, 2)'\| \geq 1$ has $y_i = 1$. Then a classifier with with (0-1 loss) $\overline{\text{err}} = 0$ is

$$\hat{f}(\mathbf{x}) = I \left[(x_{i1} - 2)^2 + (x_{i2} - 2)^2 \geq 1 \right]$$

which is, for example, not expressible in terms of two regions in \mathfrak{R}^2 with linear boundaries. However, if one defines the nonlinear transform $T: \mathfrak{R}^2 \rightarrow \mathfrak{R}^5$ by

$$T(\mathbf{x}) = (x_1, x_2, x_1^2, x_2^2, x_1 x_2)'$$

¹¹The theory of statistical sufficiency is concerned with what non-one-to-one transforms do not cause loss of information.

¹²For example, reducing a signal to a set of Fourier coefficients does not increase information about the signal. But it does replace the signal with a set of variables that are potentially more convenient than the original signal itself in terms of existing signal processing methodology.

then a very small amount of algebra shows that the classifier can be written in terms of a linear combination of coordinates of $T(\mathbf{x})$ as

$$(-4, -4, 1, 1, 0) T(\mathbf{x}) \geq -7$$

That is, thought of as defined in terms of $T(\mathbf{x}) \in \mathfrak{R}^5$ (in terms of the input transformed to the higher-dimension space \mathfrak{R}^5) the classifier is defined by a very simple linear (inner product) operation.

The toy example is instructive because it has characteristics of a strategy that is commonly effective in practice. That is one where a *nonlinear* transform is employed to map training cases into a *linear* space in which simple operations are used to define a predictor. (It should be noted that in the event that \mathbf{x} takes values in a linear space like \mathfrak{R}^2 , a *linear* transform has no potential to provide the kind of advantage seen in the hypothetical example. That is because a linear transform can only map the training set to a linear subspace of dimension no more than that spanned by the original training set.)

Notice that the thinking here substantially blurs any perceived line between "feature engineering" and "predictor fitting." They are both really parts of a single process and one cannot be treated as inconsequential to the production of the test error, Err (nor ignored it attempts to represent it empirically through cross-validation).

It is also important to think clearly about what goes into the making of transformed feature $T(\mathbf{x})$. The intent of the notation $T(\mathbf{x})$ is that the form of the function $T(\cdot)$ does not depend upon the training set. But sometimes data "pre-processing" effectively violates this understanding, making the form of the function training-set-dependent. One might use notation like $T(\mathbf{T}, \cdot)$ to represent this and this issue must be carefully handled in cross-validation.

That is, if one is contemplating use of a predictor built upon a training set $(T(\mathbf{T}, \mathbf{x}_1), y_1), (T(\mathbf{T}, \mathbf{x}_2), y_2), \dots, (T(\mathbf{T}, \mathbf{x}_N), y_N)$ and hopes to use K -fold cross-validation to reliably predict predictor performance, fitting on remainder k must be done using **not** values $(T(\mathbf{T}, \mathbf{x}_i), y_i)$ for cases in remainder k , **but rather** values $(T(\mathbf{T} - \mathbf{T}_k, \mathbf{x}_i), y_i)$. For example, as mentioned in Section 1.3.6, when building predictors based on standardized inputs, standardization must be done afresh for each new remainder! If the training set will be used to choose a parameter of a kernel for use in defining abstract features associated with input vectors, the same kind of choice must be made one remainder at a time, etc. Failure to do so breaks the cross-validation paradigm and the basic maxim that **whatever is ultimately going to be done to make predictions must be done in each individual remainder, i.e. must be done K times**. Typically, failure to follow this maxim will produce unduly optimistic (and substantially wrong) supposed "cross-validation errors."

This matter seems particularly important to recognize in cases (like those where a training set will be used to make approximate likelihood ratios per Section 1.4.2) where the *responses* in the training set or remainder (not the inputs only) are involved in the making of new features. The issue also raises the question of exactly how best to use a training set (or remainder) to both 1)

choose T as a function of the training set (or remainder) and then 2) build a predictor. Two possibilities are to 1) use the entire training set (or remainder) in both steps, or to 2) randomly split the training set (or remainder) into two parts, the first for use in choosing the form of T and the other for use in subsequently building the prediction algorithm. Which of these (or some other version of them) is likely to be most effective is not clear. What *is* clear is that care must be taken to "separately do in each remainder in a cross-validation all that will be ultimately done with the full training set" if one is to produce reliable cross-validation errors.

1.5 Some More Generalities for 2-Class Classification

In Section 1.3.2 we identified a theoretically optimal (0-1 loss) K -class classifier as

$$f(\mathbf{x}) = \arg \max_k P[y = k | \mathbf{x}]$$

By far, the most important version of this is the $K = 2$ case. And for this case, there are some very important additional general insights that we proceed to discuss.

1.5.1 More on the Form of an Optimal 0-1 Loss Classifier for $K = 2$

For $K = 2$, for various purposes different ones of the (arbitrary and completely equivalent) codings for the possible values of y

$$\{0, 1\}, \{1, 2\}, \text{ and } \{-1, 1\}$$

prove useful. For the time being, employ the first and abbreviate $P[y = 1]$ as π (so that $P[y = 0] = 1 - \pi$), and write $p(\mathbf{x}|1)$ and $p(\mathbf{x}|0)$ for the two class-conditional densities for \mathbf{x} . Then

$$P[y = 1 | \mathbf{x}] = \frac{\pi p(\mathbf{x}|1)}{\pi p(\mathbf{x}|1) + (1 - \pi) p(\mathbf{x}|0)} \quad \text{and} \quad (26)$$

$$P[y = 0 | \mathbf{x}] = \frac{(1 - \pi) p(\mathbf{x}|0)}{\pi p(\mathbf{x}|1) + (1 - \pi) p(\mathbf{x}|0)}$$

An optimal classifier is then

$$\begin{aligned} f(\mathbf{x}) &= I[P[y = 1 | \mathbf{x}] > .5] & (27) \\ &= I[P[y = 1 | \mathbf{x}] > P[y = 0 | \mathbf{x}]] \\ &= I\left[\frac{p(\mathbf{x}|1)}{p(\mathbf{x}|0)} > \frac{(1 - \pi)}{\pi}\right] \\ &= I\left[\mathcal{L}(\mathbf{x}) > \frac{(1 - \pi)}{\pi}\right] & (28) \end{aligned}$$

and one decides in favor of $y = 1$ when $P[y = 1 | \mathbf{x}]$ is large, or equivalently the likelihood ratio $\mathcal{L}(\mathbf{x})$ defined in (17) is large. Notice that this latter insight

makes connection to classical statistical theory and identifies the optimal classifier as a Neyman-Pearson test of the simple hypotheses $H_0 : y = 0$ versus $H_a : y = 1$ with "cut-point" the ratio $(1 - \pi) / \pi$.

As a slight generalization of this development, note that for $l_0 \geq 0$ and $l_1 \geq 0$ and an asymmetric loss

$$L(\hat{y}, y) = l_y I[\hat{y} \neq y]$$

an optimal classifier is

$$f(\mathbf{x}) = I \left[\mathcal{L}(\mathbf{x}) > \frac{(1 - \pi) l_0}{\pi l_1} \right]$$

In fact, for a completely general choice of four losses $L(\hat{y}, y)$ in a 2-class classification model, it is easy enough to argue that for $\Delta \equiv L(1, 0) - L(0, 0) - L(1, 1) + L(0, 1)$, $\Delta^* \equiv L(1, 0) - L(0, 0)$, and $R \equiv \Delta^* / |\Delta|$ an optimal classifier is

$$f(\mathbf{x}) = I[P[y = 1|\mathbf{x}] > R]$$

which for $R \in (0, 1)$ is

$$f(\mathbf{x}) = I \left[\mathcal{L}(\mathbf{x}) > \frac{(1 - \pi) R}{\pi(1 - R)} \right]$$

Shifting $P[y = 1]$: Effects on $P[y = 1|\mathbf{x}]$ and the Form of an Optimal 0-1 Loss Classifier An important issue in classification models is the effect of changes in π on both $P[y = 1|\mathbf{x}]$ and (optimal classifier) $f(\mathbf{x})$. There are situations, for example, in which π is very extreme (one class is rare)¹³ and it is then common practice to build a predictor using a training set made with relative frequency of $y = 1$ that is π^* , a value that is much more moderate (nearer to .5) than π . The obvious question is how to translate results for the synthetic value π^* to results for the real value π .

Relationship (26) implies that

$$P[y = 1|\mathbf{x}] = \frac{\mathcal{L}(\mathbf{x})}{\mathcal{L}(\mathbf{x}) + \frac{(1 - \pi)}{\pi}}$$

and that

$$\mathcal{L}(\mathbf{x}) = \frac{(1 - \pi)}{\pi} \left(\frac{P[y = 1|\mathbf{x}]}{1 - P[y = 1|\mathbf{x}]} \right)$$

So, for the time being subscripting P with either π or π^* depending upon which marginal probability of $y = 1$ is operating (in models with the same class-conditional densities $p(\mathbf{x}|1)$ and $p(\mathbf{x}|0)$),

$$P_\pi[y = 1|\mathbf{x}] = \frac{\frac{(1 - \pi^*)}{\pi^*} \left(\frac{P_{\pi^*}[y = 1|\mathbf{x}]}{1 - P_{\pi^*}[y = 1|\mathbf{x}]} \right)}{\frac{(1 - \pi^*)}{\pi^*} \left(\frac{P_{\pi^*}[y = 1|\mathbf{x}]}{1 - P_{\pi^*}[y = 1|\mathbf{x}]} \right) + \frac{(1 - \pi)}{\pi}} \quad (29)$$

¹³The terminology of "extreme class imbalance" is commonly used.

from which it is obvious how to translate an estimate of $P_{\pi^*}[y = 1|\mathbf{x}]$ made from a synthetically balanced training set to one for the real situation described by π . Further, an optimal classifier (27) or (28) is

$$I \left[\left(\frac{P_{\pi^*}[y = 1|\mathbf{x}]}{1 - P_{\pi^*}[y = 1|\mathbf{x}]} \right) > \frac{\pi^*(1 - \pi)}{(1 - \pi^*)\pi} \right]$$

and it is obvious how to translate an estimate of $P_{\pi^*}[y = 1|\mathbf{x}]$ made from a synthetically balanced training set to an approximately optimal classification for the real situation described by π .

For example, considering the k -nearest neighbor set-up of Section 1.3.3 using a training set made with relative frequency of $y = 1$ that is π^* when the real probability that $y = 1$ is π , the right use of a neighborhood of \mathbf{x} containing $n_1(\mathbf{x})$ cases \mathbf{x}_i with $y = 1$ and $n_0(\mathbf{x}) = k - n_1(\mathbf{x})$ with $y = 0$, is to classify according to

$$I [n_1(\mathbf{x})(1 - \pi^*)\pi > n_0(\mathbf{x})\pi^*(1 - \pi)]$$

which is the appropriate modification of the simple k -nearest neighbor rule made to account for the difference between π^* and π .

1.5.2 Other Prediction Problems in 2-Class Classification Models

There are other (besides 0-1 loss) standard prediction problems sometimes considered in the 2-class classification model. (These often show up as alternatives to use of classification error rate as test criteria in prediction contests.)

One such problem concerns class probability prediction. In a context where y is in $\{0, 1\}$ but \hat{y} is allowed to be any real number in $[0, 1]$, the so-called "log loss" (that is the 2-class version of the cross-entropy loss of Section 1.3.2 as well as the negative Bernoulli log-likelihood)

$$L(\hat{y}, y) = -y \ln \hat{y} - (1 - y) \ln (1 - \hat{y})$$

is sometimes employed. For this loss, a theoretically optimal predictor is

$$f(\mathbf{x}) = E[y|\mathbf{x}] = P[y = 1|\mathbf{x}]$$

For reasons that will shortly become clear (in Section 1.5.3), it is sometimes convenient to use not 0-1 coding but rather $-1-1$ coding in 2-class classification models, so that y is in $\{-1, 1\}$. Suppose that \hat{y} is allowed to be any real number, then three other (initially odd-looking) losses are sometimes considered, namely

$$\begin{aligned} L_1(\hat{y}, y) &= \ln(1 + \exp(-y\hat{y})) / \ln(2) \quad , \\ L_2(\hat{y}, y) &= \exp(-y\hat{y}) \quad , \text{ and} \\ L_3(\hat{y}, y) &= (1 - y\hat{y})_+ \end{aligned}$$

For these losses, theoretically optimal predictors are respectively

$$\begin{aligned} f_1(\mathbf{x}) &= \ln \left(\frac{P[y = 1|\mathbf{x}]}{P[y = -1|\mathbf{x}]} \right) = \ln \mathcal{L}(\mathbf{x}) \quad , \\ f_2(\mathbf{x}) &= \frac{1}{2} \ln \left(\frac{P[y = 1|\mathbf{x}]}{P[y = -1|\mathbf{x}]} \right) = \frac{1}{2} \ln \mathcal{L}(\mathbf{x}) \quad , \text{ and} \\ f_3(\mathbf{x}) &= \text{sign}(P[y = 1|\mathbf{x}] - P[y = -1|\mathbf{x}]) \end{aligned}$$

The "AUC" Criterion Another problem related to 2-class classification uses (1 minus) an "Area Under the Curve" (AUC) as a loss. One chooses a function $\mathcal{O}(\mathbf{x})$ taking values in $[-\infty, \infty]$ to *order* values of \mathbf{x} (large $\mathcal{O}(\mathbf{x})$ indicating large likelihood that $y = 1$). For independent \mathbf{x} with the (P) distribution of $\mathbf{x}|y = 0$ and \mathbf{x}^* with the (P) distribution of $\mathbf{x}|y = 1$ the theoretical "AUC" for \mathcal{O} (to be maximized) is

$$P[\mathcal{O}(\mathbf{x}) < \mathcal{O}(\mathbf{x}^*)] \tag{30}$$

Arguments below (based on "receiver operating characteristic curves" and Neyman-Pearson theory) establish that an optimal $\mathcal{O}(\mathbf{x})$ is the likelihood ratio $\mathcal{L}(\mathbf{x})$ defined in (17) *or* any monotone increasing transform, of it including $P[y = 1|\mathbf{x}]$.

AUC Technical Details Conceptually, an empirical "ROC" curve for a test set is this. For M test cases with M_0 actual $y_i = 0$ cases and $M_1 = M - M_0$ actual $y_i = 1$ cases, one plots M_0 points

$$\left(\frac{j}{M_0}, \hat{p}_{1j} \right) \text{ for } j = 1, 2, \dots, M_0 - 1, M_0 \tag{31}$$

where if the test cases are arranged left to right as judged least-to-most likely to have $y_i = 1$,

\hat{p}_{1j} = the fraction of $y_i = 1$ cases to the right of the j th left-most $y_i = 0$ case

If one then makes a step function from the plotted points (constant at the vertical of a plotted point over the interval of length $1/M_0$ to its left) and then computes the area under that "curve" one obtains an "AUC" (a figure of merit often used in predictive analytics contests). If the ordering of cases comes from \mathcal{O} , this area is

$$AUC = \frac{1}{M_0} \sum_{j=1}^{M_0} \hat{p}_{1j} = \frac{1}{M_0} \sum_{i \text{ s.t. } y_i=0} \left(\frac{1}{M_1} \sum_{j \text{ s.t. } y_j=1} I[\mathcal{O}(\mathbf{x}_i) < \mathcal{O}(\mathbf{x}_j)] \right) \tag{32}$$

Let G_0 and G_1 be respectively the $y = 0$ and $y = 1$ class conditional cdfs of $\mathcal{O}(\mathbf{x})$. Then corresponding to the empirical AUC is the theoretical ("integrated power") value

$$IP = \int (1 - G_1(t)) dG_0(t) \tag{33}$$

IP is exactly the criterion (30) and in the event that $G_0(t)$ is continuous and increasing (and thus has an inverse) this is

$$IP = \int_0^1 (1 - G_1(G_0^{-1}(u))) du$$

Notice too that if for each t one builds from \mathcal{O} a classifier of the form

$$a_t(\mathbf{x}) = I[\mathcal{O}(\mathbf{x}) > t]$$

the integrand in display (33) is the power of the test/classifier as a function of t and IP is an average (according to the $y = 0$ class conditional distribution of $\mathcal{O}(\mathbf{x})$), an "integrated power."

Let $\alpha(t)$ be the Type I error rate of the test $a_t(\mathbf{x})$, i.e.

$$\alpha(t) = E_0 a_t(\mathbf{x}) = P[\mathcal{O}(\mathbf{x}) > t | y = 0]$$

and $\beta(t)$ be the Type II error rate of $a_t(\mathbf{x})$,

$$\beta(t) = 1 - E_1 a_t(\mathbf{x}) = P[\mathcal{O}(\mathbf{x}) \leq t | y = 1]$$

Another representation of IP is then this. As t runs from ∞ to $-\infty$ the points

$$(\alpha(t), 1 - \beta(t)) \tag{34}$$

trace out a (theoretical Receiver Operating Characteristic) curve in $[0, 1]^2$ (the theoretical version of the step function defined by points in display (31) made from ordered test cases in order to compute the empirical AUC). The ordinary integral over $[0, 1]$ of the function defined by that parametric curve is IP , and therefore the "higher" that parametric curve, the larger is the (theoretical) IP .

But consider the convex body in $[0, 1]^2$ defined by all pairs $(\alpha, 1 - \beta)$ corresponding to possible classifiers/tests (we may need to allow randomization here). (This is a reflection of the set of all points (α, β) comprising the 0-1 loss risk set of all possible classifiers/tests.) The upper boundary of that convex body (that corresponds to the lower boundary of the risk set) comes from Bayes classifiers/tests. It is guaranteed to lie "above" (at least as high) as the parametric curve defined in display (34). But the form of optimal (Bayes and Neyman-Pearson) tests/classifiers is well-known. We have already said that an optimal classifier in the present context is as in display (28) and this brings us to the conclusion: any \mathcal{O} that is a monotone increasing transformation of the likelihood ratio $\mathcal{L}(\mathbf{x})$ (is equivalent to the likelihood ratio) will optimize IP .

1.5.3 "Voting Functions," Losses for Them, and Expected 0-1 Loss

The fact that empirical search for a good 2-class classifier is essentially search for a good approximation to the likelihood ratio function $\mathcal{L}(\mathbf{x})$ raises another kind of consideration for 2-class problems. That is the possibility of focusing on the building of a good "voting function" $g(\mathbf{x})$ to underlie a classifier.

For the time being, it's now convenient to employ the $-1-1$ coding of class labels (use $\mathcal{G} = \{-1, 1\}$) and to without much loss of generality consider classifiers defined for an arbitrary voting function $g(\mathbf{x})$ by

$$f(\mathbf{x}) = \text{sign}(g(\mathbf{x}))$$

(except for the possibility that $g(\mathbf{x}) = 0$, that typically has 0 probability for both classes). Then an optimal voting function for 0-1 loss is

$$g^{\text{opt}}(\mathbf{x}) = \frac{p(\mathbf{x}|1)}{p(\mathbf{x}|-1)} - \frac{P[y = -1]}{P[y = 1]} \quad (35)$$

With this notation, a classifier $f(\mathbf{x}) = \text{sign}(g(\mathbf{x}))$ produces 0-1 loss neatly written as

$$L(\hat{y}, y) = I[yg(\mathbf{x}) < 0]$$

(a loss of 1 is incurred when y and $g(\mathbf{x})$ have opposite signs). So the the 0-1 loss expected loss/error rate has the useful representation

$$EI[yg(\mathbf{x}) < 0] \quad (36)$$

We have seen that a function g optimizing the average value (36) is $g^{\text{opt}}(\mathbf{x})$ defined in (35). But the indicator function $I[u < 0]$ involved in (36) is discontinuous (and thus non-differentiable), and for some purposes it would be more convenient to work with a continuous (even differentiable) one in making an empirical choice of voting function.

If $I[u < 0] \leq h(u)$, it is obvious that

$$EI[yg(\mathbf{x}) < 0] \leq Eh(yg(\mathbf{x})) \quad (37)$$

So the right hand side of display (37) functions as an upper bound for the 0-1 loss error rate and an approximate (data-based) minimizer of that right hand side used as a voting function can be expected to control 0-1 loss error rate. Several different continuous choices of "loss" $h(u)$ can be viewed as motivating popular methods of (voting function and) classifier development. These include:

1. $h_1(u) = \ln(1 + \exp(-u)) / \ln(2)$ a function related to a Bernoulli negative log-likelihood term when $yg(\mathbf{x})$ is substituted for u ,
2. $h_2(u) = \exp(-u)$ (the "exponential loss") associated with the AdaBoost.M1 algorithm, and
3. $h_3(u) = (1 - u)_+$ (the "hinge loss") associated with "support vector machines."

For reference, the indicator function $I[u < 0]$ and the functions $h_1(u)$, $h_2(u)$, and $h_3(u)$ are plotted together in Figure 4.

One reason why this line of argument proves effective is that not only does bound (37) hold, but minimizers of a $Eh(yg(\mathbf{x}))$ over choice of function g for

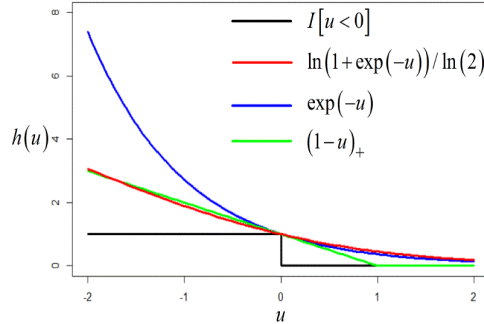


Figure 4: "Losses" $I[u < 0]$ in black, $h_1(u)$ in red, $h_2(u)$ in blue, and $h_3(u)$ in green.

standard choices of h with $h(u) \geq I[u < 0]$ are directly related to the likelihood ratio. This can be seen using the results concerning optimal predictors in 2-class classification models from Section 1.5.2. That is,

$$\begin{aligned}
 Eh_1(yg(\mathbf{x})) = EL_1(g(\mathbf{x}), y) & \text{ has optimizer } g_1^{\text{opt}}(\mathbf{x}) = \ln\left(\frac{P[y = 1|\mathbf{x}]}{P[y = -1|\mathbf{x}]}\right) \\
 Eh_2(yg(\mathbf{x})) = EL_2(g(\mathbf{x}), y) & \text{ has optimizer } g_2^{\text{opt}}(\mathbf{x}) = \frac{1}{2} \ln\left(\frac{P[y = 1|\mathbf{x}]}{P[y = -1|\mathbf{x}]}\right) \text{ and} \\
 Eh_3(yg(\mathbf{x})) = EL_3(g(\mathbf{x}), y) & \text{ has optimizer } g_3^{\text{opt}}(\mathbf{x}) = \text{sign}\left(\frac{P[y = 1|\mathbf{x}]}{P[y = -1|\mathbf{x}]} - 1\right)
 \end{aligned}$$

The first two functions are monotone transformations of the likelihood ratio and when used as a voting function produce a (0-1 loss) optimal classifier. The third is *the optimal classifier itself*.

So empirical search for optimizers of (an empirical version of) the risk $Eh(yg(\mathbf{x}))$ can produce good classifiers. This has the fascinating effect of making SEL prediction and classification look very much alike. Ultimately, in development of a predictor, one is searching among some class of functions, \mathcal{S} , for a *real-valued* g making an appropriate empirical approximation of a risk measure small.

1.6 Density Estimation and Approximately Optimal and Naive Bayes Classification

As another preliminary, we make a few comments on the problem of density estimation. One might phrase the problem of describing structure for \mathbf{x} in terms of estimating a pdf for the variable. And a naive way of approximating a theoretically optimal classifier might be to directly estimate both class probabilities and class conditional densities and use them in place of their estimands

in the optimal form (2) to produce

$$\hat{f}(\mathbf{x}) = \arg \max_k P[\widehat{y} = k] p(\widehat{\mathbf{x}}|k) \quad (38)$$

So we consider the problem:

given $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ iid with (unknown) pdf $q(\mathbf{x})$, how to estimate q ?

Initially suppose that $p = 1$. For $g(\cdot)$ some fixed pdf (like, for example, the standard normal pdf), invent a location-scale family of densities on \mathfrak{R} by defining (for "bandwidth" $\lambda > 0$)

$$h(\cdot|\theta, \lambda) = \frac{1}{\lambda} g\left(\frac{\cdot - \theta}{\lambda}\right)$$

One may think of a corresponding "kernel" (this is a potentially different usage of the word "kernel" than that in Section 1.4.3 and no non-negative definiteness of the function is needed or assumed)

$$\mathcal{K}_\lambda(\cdot, \theta) \equiv g\left(\frac{\cdot - \theta}{\lambda}\right)$$

The Parzen estimate of $q(x_0)$ is then

$$\begin{aligned} \hat{q}_\lambda(x_0) &= \frac{1}{N} \sum_{i=1}^N h(x_0|x_i, \lambda) \\ &= \frac{1}{\lambda N} \sum_{i=1}^N \mathcal{K}_\lambda(x_0, x_i) \end{aligned}$$

an average of kernel values.

A standard choice of univariate density $g(\cdot)$ is $\phi(\cdot)$, the standard normal pdf. A way to think about the density estimate that results from using a normal kernel is as representing the distribution of "a random choice from the training set perturbed by a mean 0 normal error with standard deviation equal to the bandwidth." If the bandwidth is extremely small, the density estimate will essentially consist of "spikes" at the x_i in the training set. If it is extremely large, the density estimate will essentially consist of a normal density centered around the mean of the x_i . Useful bandwidths will be neither extremely small nor extremely large.

Figure 5 provides a $p = 1$ pdf $q(x)$ (in black), a sample of size $N = 100$ from the distribution and three Parzen estimates of q made with $g(\cdot) = \phi(\cdot)$ and bandwidths $\lambda = .2$ (red), .4 (blue), and .5 (green).

The natural generalization of this to p dimensions is to use a MVN_p density as a kernel (scaled according to λ). One should expect that unless N is huge, this methodology will be reliable only for fairly small p (say 3 at most) as a means of estimating a general p -dimensional pdf. Figure 6 provides two representations

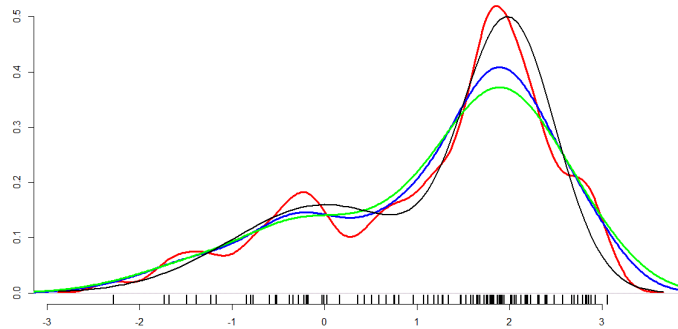


Figure 5: A $p = 1$ density, a corresponding sample of $N = 100$ values x , and three density estimates based on different bandwidths.

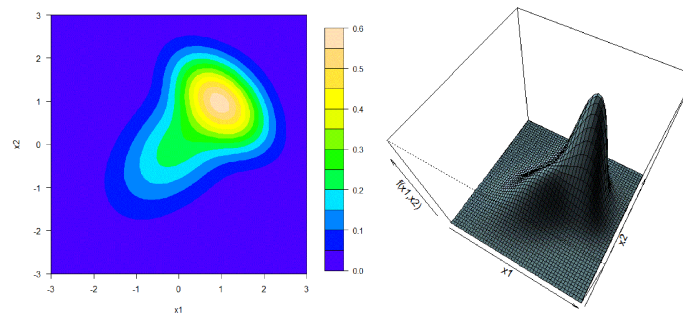


Figure 6: Two representations of a particular 2-d pdf (a mixture of two bivariate normal densities).

of a bivariate density. Figure 7 then shows several samples of size $N = 100$ from the density and corresponding bivariate kernel density estimates made with software-default choices of multivariate bandwidth covariance matrices.

In the event that N is big, dimension p is low, and K is small, one might at least consider estimating the densities $p(\mathbf{x}|k)$ (with, say, kernel density estimates $\widehat{p(\mathbf{x}|k)}$) using relative frequencies of values of y (say $P[\widehat{y = k}]$) to estimate the probabilities $P[y = k]$, and employing a classifier like the one in display (38). Figure 8 shows for a sample of size $N = 100$ from both the bivariate density of Figure 6 and a uniform density on $[-3, 3]^2$ density estimates and their ratio (an approximate likelihood ratio). Since the denominator density is uniform the actual likelihood ratio is equivalent to the density portrayed in Figure 6 and it seems like (for this sample at least) an approximation to a Bayes classifier based on density estimates might work reasonably well in this particular problem.

It is worth considering the form that such estimated-density-approximately-Bayes classifiers take in the case where symmetric Gaussian kernels are used.

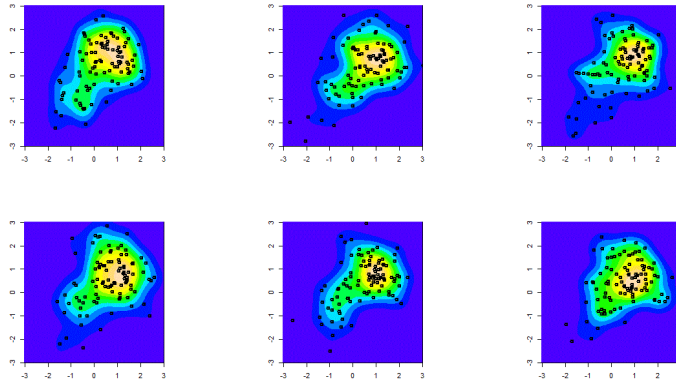


Figure 7: 6 samples of size $N = 100$ from the bivariate density of Figure 6 and density estimates made using the `kde2d` function in the MASS package with default choice of "bandwidth" covariance matrix.

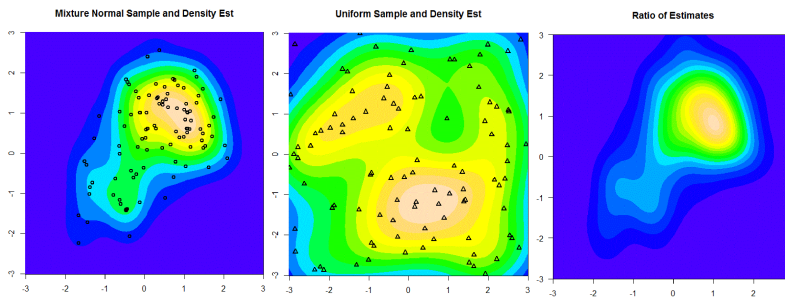


Figure 8: Two $N = 100$ density estimates and their ratio for classifications between Uniform $[-3, 3]^2$ and the distribution of Figure 6.

That is, consider the case where one uses as a multivariate density estimate

$$\widehat{q}(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^N \phi(\mathbf{x} | \mathbf{x}_i, \lambda^2 \mathbf{I})$$

(where $\phi(\cdot | \boldsymbol{\theta}, \boldsymbol{\Sigma})$ is the MVN_p density with mean vector $\boldsymbol{\theta}$ and covariance matrix $\boldsymbol{\Sigma}$). A bit of algebra shows that with this kind of multivariate estimates of class-conditional densities (based on the parts of the training set with $y = k$) (and using training set relative frequencies to estimate class probabilities) the approximately Bayes classifier is

$$\hat{f}_\lambda(\mathbf{x}) = \arg \max_k \sum_{i \text{ s.t. } y_i = k} \exp\left(-\frac{1}{2\lambda^2} \|\mathbf{x} - \mathbf{x}_i\|^2\right)$$

This is a plausible kind of form, classifying to class k when \mathbf{x} is "close to"

relatively many training inputs from class k . The bandwidth might be chosen based on cross-validation of classifier performance.

The statistical folklore is that this kind of classifier can work poorly in high dimensions because of the imprecisions (large variances) of the density estimators. The "estimated density" approximations to the optimal rule are based on what are usually low-bias-but-high-variance estimators. As such, the corresponding classifiers are very flexible, but can perform poorly for small training sets. Less flexible classification methods will often perform much better in practical problems (although those methods may be incapable of approximating the optimal rule for all cases, even if N is huge).

There is a variant of form (38) that is thought to sometimes be effective even when p is not small (and p -dimensional density estimation is hopeless). The basic idea is to estimate 1-dimensional marginals of the $p(\mathbf{x}|k)$ s and use their products in place of $p(\mathbf{x}|k)$ s. That is, if for each k the density $p(\mathbf{x}|k) : \mathcal{R}^p \rightarrow \mathcal{R}^+$ has marginal densities $p_1(x_1|k), p_2(x_2|k), \dots, p_p(x_p|k)$ (each mapping $\mathcal{R} \rightarrow \mathcal{R}^+$), while it may not be feasible to estimate $p(\mathbf{x}|k)$, it could be possible to effectively estimate $p_1(x_1|k), p_2(x_2|k), \dots, p_p(x_p|k)$. If this is the case, the classifier

$$\hat{f}(\mathbf{x}) = \arg \max_k P[y = k] \prod_{j=1}^p \widehat{p}_j(x_j|k)$$

might be employed. (That is, one might treat elements x_j of \mathbf{x} as if they were independent for every k , and multiply together kernel estimates of marginal densities.) This has been called a "**naïve Bayes**" classifier.

The method seems to have a reputation for often being useful. But there will certainly be situations where it doesn't work very well because of failure to account for strong dependencies between input variables. Figure 9 shows the common marginal for x_1 and x_2 corresponding to the distribution of Figure 6. Figure 10 then shows the original bivariate density and the distribution of independence with the same marginals. The product density is clearly quite different from the original and estimation of the marginals alone can at best only reproduce the product form.

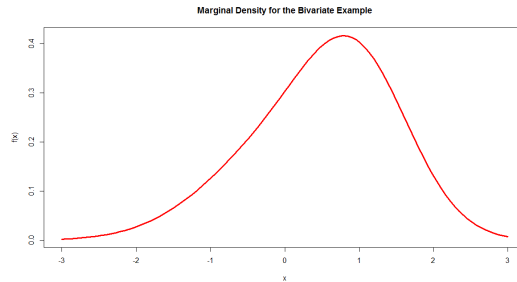


Figure 9: The common marginal pdf for both x_1 and x_2 for the bivariate distribution of Figure 6.

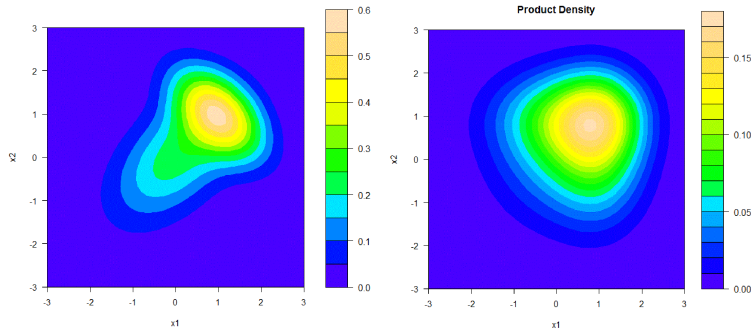


Figure 10: Original bivariate density from Figure 6 and a product density based on the marginal(s) (as pictured in Figure 9).

1.7 Plotting to Portray the Effects of Particular Inputs in Prediction

An issue briefly discussed in HTF Ch 10 is the making and plotting of functions a few of the coordinates of \mathbf{x} in an attempt to understand the nature of the influence of these in a predictor. (What they say is really perfectly general, not at all special to the particular form of predictors discussed in that chapter.) If, for example, I want to understand the influence the first two coordinates of \mathbf{x} have on a form $f(\mathbf{x})$, I might think of somehow averaging out the remaining coordinates of \mathbf{x} . One theoretical implementation of this idea would be

$$\bar{f}_{12}(x_1, x_2) = \mathbb{E}^{(x_3, x_4, \dots, x_p)} f(x_1, x_2, x_3, x_4, \dots, x_p)$$

i.e. averaging according to the marginal distribution of the excluded input variables. An empirical version of this is

$$\frac{1}{N} \sum_{i=1}^N f(x_1, x_2, x_{3i}, x_{4i}, \dots, x_{pi})$$

This might be plotted (e.g. in contour plot fashion) and the plot called a partial dependence plot for the variables x_1 and x_2 . HTF's language is that this function details the dependence of the predictor on (x_1, x_2) "after accounting for the average effects of the other variables." This thinking amounts to a version of the kind of thing one does in ordinary factorial linear models, where main effects are defined in terms of average (across all levels of all other factors) means for individual levels of a factor, two-factor interactions are defined in terms of average (again across all levels of all other factors) means for pairs of levels of two factors, etc.

Something different raised by HTF is consideration of

$$\tilde{f}_{12}(x_1, x_2) = \mathbb{E}[f(\mathbf{x}) | x_1, x_2]$$

(This is, by the way, the function of (x_1, x_2) closest to $f(\mathbf{x})$ in $L_2(P)$.) This is obtained by averaging not against the marginal of the excluded variables, but against the conditional distribution of the excluded variables given x_1 and x_2 . No workable empirical version of $\tilde{f}_{12}(x_1, x_2)$ can typically be defined. And it should be clear that this is *not* the same as $\bar{f}_{12}(x_1, x_2)$. HTF say this in some sense describes the effects of (x_1, x_2) on the prediction "ignoring the impact of the other variables." (In fact, if $f(\mathbf{x})$ is a good approximation for $E[y|\mathbf{x}]$, this conditioning produces essentially $E[y|x_1, x_2]$ and \tilde{f}_{12} is just a predictor of y based on (x_1, x_2) .)

The difference between \bar{f}_{12} and \tilde{f}_{12} is easily seen through resort to a simple example. If, for example, f is additive of the form

$$f(\mathbf{x}) = h_1(x_1, x_2) + h_2(x_3, \dots, x_p)$$

then

$$\begin{aligned}\bar{f}_{12}(x_1, x_2) &= h_1(x_1, x_2) + E h_2(x_3, \dots, x_p) \\ &= h_1(x_1, x_2) + \text{constant}\end{aligned}$$

while

$$\tilde{f}_{12}(x_1, x_2) = h_1(x_1, x_2) + E[h_2(x_3, \dots, x_p) | x_1, x_2]$$

and the \tilde{f}_{12} "correction" to $h_1(x_1, x_2)$ is not necessarily constant in (x_1, x_2) .

The upshot of all this is that partial dependence plots are potentially helpful, but that one needs to remember that they are produced by averaging according to the marginal of the set of variables not under consideration.

2 Some Linear Theory, Linear Algebra, and Principal Components

Methods of modern multivariate statistical learning often involve more background in the theory of linear spaces and linear algebra than is assumed or used in a basic linear models course. So here we provide some of that and then apply it to the unsupervised learning problem of "principal components analysis."

2.1 Inner Product Spaces

Most of applied mathematics in general and statistical machine learning in particular is built on the notions of "linear combinations" of various objects and "inner products" of these (that in turn lead to coherent notions of their "sizes" and of "distances" between them). Here we briefly review what is necessary for a theory of such objects and operations to make sense.

First, a vector (or linear) space \mathbf{V} consists of objects $\mathbf{v}, \mathbf{w}, \dots$ such that if $\mathbf{v} \in \mathbf{V}$ and $a \in \mathfrak{R}$, then the object $a\mathbf{v}$ makes sense and belongs to \mathbf{V} , and for \mathbf{v} and \mathbf{w} in \mathbf{V} the object $\mathbf{v} + \mathbf{w}$ also makes sense and belongs to \mathbf{V} . The archetypal

vector spaces are the Euclidean spaces \Re^p where elements are "ordinary" p -dimensional vectors. But other kinds of vector spaces are useful in statistical machine learning as well, including function spaces. Take for example the set of functions on $[0, 1]$ that have finite integrals of their squares. (This space is sometimes known as $L_2([0, 1])$.) More or less obviously, if $g : [0, 1] \rightarrow \Re$ with $\int_0^1 (g(x))^2 dx < \infty$ and $a \in \Re$, then $ag(x)$ makes sense, maps $[0, 1]$ to \Re and has $\int_0^1 (ag(x))^2 dx = a^2 \int_0^1 (g(x))^2 dx < \infty$. Further, if $g : [0, 1] \rightarrow \Re$ with $\int_0^1 (g(x))^2 dx < \infty$ and $h : [0, 1] \rightarrow \Re$ with $\int_0^1 (h(x))^2 dx < \infty$, then the function $g(x) + h(x)$ makes sense, maps $[0, 1]$ to \Re and has finite integral of its square.

The notion of an inner product (of pairs of elements of a vector space \mathbf{V}) is that of a symmetric (bi-)linear positive definite function $\langle \mathbf{v}, \mathbf{w} \rangle$ mapping $\mathbf{V} \times \mathbf{V} \rightarrow \Re$. That is, $\langle \mathbf{v}, \mathbf{w} \rangle$ is an inner product on the vector space \mathbf{V} if it satisfies

1. $\langle \mathbf{w}, \mathbf{v} \rangle = \langle \mathbf{v}, \mathbf{w} \rangle \forall \mathbf{v}, \mathbf{w} \in \mathbf{V}$ (symmetry),
2. $\langle a\mathbf{v}, \mathbf{w} \rangle = a \langle \mathbf{v}, \mathbf{w} \rangle \forall \mathbf{v}, \mathbf{w} \in \mathbf{V}$ and $a \in \Re$,
 $\langle \mathbf{v} + \mathbf{u}, \mathbf{w} \rangle = \langle \mathbf{v}, \mathbf{w} \rangle + \langle \mathbf{u}, \mathbf{w} \rangle \forall \mathbf{u}, \mathbf{v}$, and $\mathbf{w} \in \mathbf{V}$ (bilinearity), and
3. $\langle \mathbf{v}, \mathbf{v} \rangle \geq 0 \forall \mathbf{v} \in \mathbf{V}$ and $\langle \mathbf{v}, \mathbf{v} \rangle = 0$ if and only if $\mathbf{v} = \mathbf{0}$ (positive definiteness).

Of course Euclidean p -space is a vector space with inner product defined as the "dot-product" of p -dimensional vectors \mathbf{v} and \mathbf{w} , namely

$$\langle \mathbf{v}, \mathbf{w} \rangle = \mathbf{v}'\mathbf{w} = \sum_{j=1}^p v_j w_j$$

It is possible to argue that in the case of the $L_2([0, 1])$ function space, the integral of the product of two elements provides a valid inner product, that is

$$\langle g, h \rangle \equiv \int_0^1 g(x) h(x) dx$$

satisfies 1. through 3.

An inner product on a vector space \mathbf{V} leads immediately to notions of size and distance in the space. The norm (i.e. the "size" or "length") of an element of \mathbf{V} can be coherently defined as

$$\|\mathbf{v}\| \equiv \sqrt{\langle \mathbf{v}, \mathbf{v} \rangle}$$

Then the distance between two elements of \mathbf{V} can be taken to be the size of the difference between them. That is, the distance between \mathbf{v} and \mathbf{w} belonging to \mathbf{V} (say $d(\mathbf{v}, \mathbf{w})$) derived from the inner product is

$$d(\mathbf{v}, \mathbf{w}) = \|\mathbf{v} - \mathbf{w}\|$$

This satisfies all the properties necessary to qualify as a "metric" or "distance function," including the important triangle inequality.

In Euclidean p -space, the norm is the geometrical length of a p -vector (the root of the sum of the p squared entries of the vector) and the associated distance is ordinary Euclidean distance. In the case of the L_2 $([0, 1])$ function space, the norm/size of an element g is

$$\|g\| = \sqrt{\int_0^1 (g(x))^2 dx}$$

and the distance between elements g and h is

$$d(g, h) = \sqrt{\int_0^1 (g(x) - h(x))^2 dx}$$

Many other useful notions commonly understood in Euclidean spaces generalize directly to more abstract vector spaces and inner product spaces. \mathbf{v} and $\mathbf{w} \in \mathbf{V}$ are perpendicular or orthogonal when $\langle \mathbf{v}, \mathbf{w} \rangle = 0$. Subspaces of \mathbf{V} can be generated as all linear combinations of a set of elements of \mathbf{V} and are commonly referred to as the "span" of the set of elements. A basis for a subspace of \mathbf{V} is a set of linearly independent vectors (no linear combination of them is the $\mathbf{0}$ vector) that span the subspace. "Orthonormal" bases (whose elements are perpendicular and each of norm 1) for \mathbf{V} (or for subspaces of \mathbf{V}) are particularly attractive, as they provide very simple representations for "projections" of $\mathbf{v} \in \mathbf{V}$ onto the span of any set of them, as a linear combination of basis vectors where coefficients are the inner products with the corresponding basis vectors. In the context of machine learning, projections of a vector \mathbf{v} are very usefully thought of as "low-dimensional" approximations to \mathbf{v} (in terms of a "few" basis vectors). (The dimension of a subspace of \mathbf{V} is, just as in ordinary Euclidean spaces, the number of vectors in a basis for it.) Geometry of Euclidean cases (where subspaces are geometrical hyperplanes containing the origin and geometrical hyperplanes are subspaces potentially shifted from the origin by addition of a vector not in the subspace) is helpful in interpreting statistical machine learning constructs in more abstract inner product spaces.

2.2 The (General) Gram-Schmidt Process and the QR Decomposition of a $rank = p$ Matrix \mathbf{X}

We continue to use the notation

$$\mathbf{X}_{N \times p} = \begin{pmatrix} \mathbf{x}'_1 \\ \mathbf{x}'_2 \\ \vdots \\ \mathbf{x}'_N \end{pmatrix} \text{ and potentially } \mathbf{Y}_{N \times 1} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

and recall that it is standard linear models fare that ordinary least squares projects \mathbf{Y} onto $C(\mathbf{X})$, the column space of \mathbf{X} , in order to produce the vector

of fitted values

$$\hat{\mathbf{Y}}_{N \times 1} = \begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_N \end{pmatrix}$$

For many purposes it would be convenient if the columns of a full rank ($rank = p$) matrix \mathbf{X} were orthogonal. In fact, it would be useful to replace the $N \times p$ matrix \mathbf{X} with an $N \times p$ matrix \mathbf{Z} with orthogonal columns and having the property that for each l if \mathbf{X}_l and \mathbf{Z}_l are $N \times l$ consisting of the first l columns of respectively \mathbf{X} and \mathbf{Z} , then $C(\mathbf{Z}_l) = C(\mathbf{X}_l)$. Such a matrix can in fact be constructed using the so-called Gram-Schmidt process. This process generalizes beyond the present application to \mathfrak{R}^N to general inner product spaces, and in recognition of that important fact we'll first describe it in general terms and then consider its implications for a ($rank = p$) matrix \mathbf{X} .

Consider p vectors $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_p$ (that could be N -vectors where \mathbf{x}_j is the j th column of \mathbf{X})¹⁴. The **Gram-Schmidt process** proceeds as follows:

1. Set

$$\mathbf{z}_1 = \mathbf{x}_1 \quad \text{and} \quad \mathbf{q}_1 = \langle \mathbf{z}_1, \mathbf{z}_1 \rangle^{-1/2} \mathbf{z}_1 = \frac{1}{\|\mathbf{z}_1\|} \mathbf{z}_1$$

2. Having constructed $\{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_{l-1}\}$, let

$$\mathbf{z}_l = \mathbf{x}_l - \sum_{j=1}^{l-1} \frac{\langle \mathbf{x}_l, \mathbf{z}_j \rangle}{\langle \mathbf{z}_j, \mathbf{z}_j \rangle} \mathbf{z}_j = \mathbf{x}_l - \sum_{j=1}^{l-1} \langle \mathbf{x}_l, \mathbf{q}_j \rangle \mathbf{q}_j \quad \text{and} \quad \mathbf{q}_l = \frac{1}{\|\mathbf{z}_l\|} \mathbf{z}_l$$

Figure 11 illustrates this construction for a simple case of $p = 2$. \mathbf{z}_l is the part of \mathbf{x}_l that "sticks out of the subspace spanned by $\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_{l-1}$ " the difference between \mathbf{x}_l and the perpendicular projection of that vector onto the subspace. \mathbf{q}_l is the normalized version of \mathbf{z}_l , the unit vector pointing in the same direction as \mathbf{z}_l .

It is easy enough to see that $\langle \mathbf{z}_l, \mathbf{z}_j \rangle = 0$ for all $j < l$ (building up the orthogonality of $\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_{l-1}$ by induction), since

$$\langle \mathbf{z}_l, \mathbf{z}_j \rangle = \langle \mathbf{x}_l, \mathbf{z}_j \rangle - \langle \mathbf{x}_l, \mathbf{z}_j \rangle$$

as at most one term of the sum in step 2. above is non-zero. Further, assume that the span of $\{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_{l-1}\}$ is the same as the span of $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{l-1}\}$. \mathbf{z}_l is in the span of $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_l\}$ so that the span of $\{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_l\}$ is a subset of the span of $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_l\}$. And since any element of the span of $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_l\}$ can be written as a linear combination of an element of the span of $\{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_{l-1}\}$ (span of $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{l-1}\}$) and \mathbf{x}_l we also have that the

¹⁴Notice that this is in potential conflict with earlier notation that made \mathbf{x}_i the p -vector of inputs for the i th case in the training data. We will simply have to read the following in context and keep in mind the local convention.

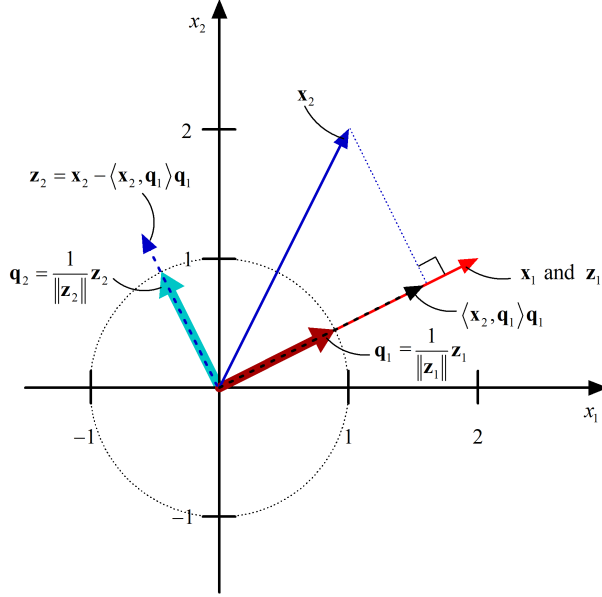


Figure 11: A $p = 2$ illustration of the Gram-Schmidt construction of an orthonormal basis for the subspace spanned by \mathbf{x}_1 and \mathbf{x}_2 .

span of $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_l\}$ is a subset of the span of $\{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_l\}$. That is that $\{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_l\}$ and $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_l\}$ have the same span and the set of vectors

$$\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_l$$

form an orthonormal basis for the span of $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_l\}$.

Since the \mathbf{z}_j are perpendicular, for any vector \mathbf{w} ,

$$\sum_{j=1}^l \frac{\langle \mathbf{w}, \mathbf{z}_j \rangle}{\langle \mathbf{z}_j, \mathbf{z}_j \rangle} \mathbf{z}_j = \sum_{j=1}^l \langle \mathbf{w}, \mathbf{q}_j \rangle \mathbf{q}_j \quad (39)$$

is the projection of \mathbf{w} onto the span of $\{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_l\}$ (of $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_l\}$). (To see this, consider minimization of the quantity $\langle \mathbf{w} - \sum_{j=1}^l c_j \mathbf{z}_j, \mathbf{w} - \sum_{j=1}^l c_j \mathbf{z}_j \rangle = \left\| \mathbf{w} - \sum_{j=1}^l c_j \mathbf{z}_j \right\|^2$ by choice of the constants c_j .) In particular, $\sum_{j=1}^{l-1} \langle \mathbf{x}_l, \mathbf{q}_j \rangle \mathbf{q}_j$ in step 2. of the Gram-Schmidt process is the projection of \mathbf{x}_l onto the span of $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{l-1}\}$.

Consider now the case where Euclidean N -vectors $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_p$ are the columns of a data matrix \mathbf{X} . Spans are column spaces of matrices. Indeed the $N \times p$ matrix \mathbf{Z} has orthogonal columns and the property that $C(\mathbf{Z}_l) = C(\mathbf{X}_l)$. And the set of vectors $\{\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_l\}$ is an orthonormal basis for this column

space. So the projection of a vector of outputs \mathbf{Y} onto $C(\mathbf{X}_l)$ is

$$\sum_{j=1}^l \frac{\langle \mathbf{Y}, \mathbf{z}_j \rangle}{\langle \mathbf{z}_j, \mathbf{z}_j \rangle} \mathbf{z}_j = \sum_{j=1}^l \langle \mathbf{Y}, \mathbf{q}_j \rangle \mathbf{q}_j$$

This means that for a full p -variable regression,

$$\frac{\langle \mathbf{Y}, \mathbf{z}_p \rangle}{\langle \mathbf{z}_p, \mathbf{z}_p \rangle}$$

is the regression coefficient for \mathbf{z}_p and (since only \mathbf{z}_p involves it) the last variable in \mathbf{X} , \mathbf{x}_p . So, in constructing a vector of fitted values, fitted regression coefficients in multiple regression can be interpreted as weights to be applied to that part of the input vector that remains *after* projecting the predictor onto the space spanned by all the others.

The construction of the orthogonal variables \mathbf{z}_j can be represented in matrix form as

$$\mathbf{X}_{N \times p} = \mathbf{Z}_{N \times p} \mathbf{\Gamma}_{p \times p}$$

where $\mathbf{\Gamma}$ is upper triangular with

$$\begin{aligned} \gamma_{kj} &= \text{the value in the } k\text{th row and } j\text{th column of } \mathbf{\Gamma} \\ &= \begin{cases} 1 & \text{if } j = k \\ \frac{\langle \mathbf{z}_k, \mathbf{x}_j \rangle}{\langle \mathbf{z}_k, \mathbf{z}_k \rangle} & \text{if } j > k \end{cases} \end{aligned}$$

Defining

$$\mathbf{D} = \mathbf{diag}(\langle \mathbf{z}_1, \mathbf{z}_1 \rangle^{1/2}, \dots, \langle \mathbf{z}_p, \mathbf{z}_p \rangle^{1/2}) = \mathbf{diag}(\|\mathbf{z}_1\|, \dots, \|\mathbf{z}_p\|)$$

and letting

$$\mathbf{Q} = \mathbf{Z}\mathbf{D}^{-1} \quad \text{and} \quad \mathbf{R} = \mathbf{D}\mathbf{\Gamma}$$

one may write

$$\mathbf{X} = \mathbf{Q}\mathbf{R} \tag{40}$$

that is the so-called QR decomposition of \mathbf{X} .

Note that the notation used here is consistent, in that for \mathbf{q}_j the j th column of \mathbf{Q} , $\mathbf{q}_j = (\langle \mathbf{z}_j, \mathbf{z}_j \rangle)^{-1/2} \mathbf{z}_j$ as was used in defining the Gram-Schmidt process. In display (40), \mathbf{Q} is $N \times p$ with

$$\mathbf{Q}'\mathbf{Q} = \mathbf{D}^{-1}\mathbf{Z}'\mathbf{Z}\mathbf{D}^{-1} = \mathbf{D}^{-1} \mathbf{diag}(\langle \mathbf{z}_1, \mathbf{z}_1 \rangle, \dots, \langle \mathbf{z}_p, \mathbf{z}_p \rangle) \mathbf{D}^{-1} = \mathbf{I}$$

consistent with the fact that \mathbf{Q} has for columns perpendicular unit vectors that form a basis for $C(\mathbf{X})$. \mathbf{R} is upper triangular and that says that only the first l of these unit vectors are needed to create \mathbf{x}_l .

The decomposition is computationally useful in that the projection of a response vector \mathbf{Y} onto $C(\mathbf{X})$ is

$$\hat{\mathbf{Y}} = \sum_{j=1}^p \langle \mathbf{Y}, \mathbf{q}_j \rangle \mathbf{q}_j = \mathbf{Q}\mathbf{Q}'\mathbf{Y} \quad (41)$$

and

$$\hat{\boldsymbol{\beta}}^{\text{ols}} = \mathbf{R}^{-1}\mathbf{Q}'\mathbf{Y}$$

(The fact that \mathbf{R} is upper triangular implies that there are efficient ways to compute its inverse.)

2.3 The Singular Value Decomposition of \mathbf{X}

If the $N \times p$ matrix \mathbf{X} has rank r then it has a so-called singular value decomposition as

$$\mathbf{X} = \underset{N \times p}{\mathbf{U}} \underset{N \times r}{\mathbf{D}} \underset{r \times p}{\mathbf{V}'}$$

where \mathbf{U} has orthonormal columns (left singular vectors) spanning $C(\mathbf{X})$, \mathbf{V} has orthonormal columns (right singular vectors) spanning $C(\mathbf{X}')$ (the row space of \mathbf{X}), and $\mathbf{D} = \mathbf{diag}(d_1, d_2, \dots, d_r)$ for

$$d_1 \geq d_2 \geq \dots \geq d_r > 0$$

the d_j are the "singular values" of \mathbf{X} .¹⁵

An interesting property of the singular value decomposition is this. If \mathbf{U}_l and \mathbf{V}_l are matrices consisting of the first $l \leq r$ columns of \mathbf{U} and \mathbf{V} respectively, then

$$\mathbf{X}^{*l} = \mathbf{U}_l \mathbf{diag}(d_1, d_2, \dots, d_l) \mathbf{V}_l'$$

is the best (in the sense of squared distance from \mathbf{X} in \Re^{Np}) $rank = l$ approximation to \mathbf{X} . (Note that application of this kind of argument to covariance matrices provides low-rank approximations to complicated covariance matrices.)

Since the columns of \mathbf{U} are an orthonormal basis for $C(\mathbf{X})$, the projection of an output vector \mathbf{Y} onto $C(\mathbf{X})$ is

$$\hat{\mathbf{Y}}^{\text{ols}} = \sum_{j=1}^r \langle \mathbf{Y}, \mathbf{u}_j \rangle \mathbf{u}_j = \mathbf{U}\mathbf{U}'\mathbf{Y} \quad (42)$$

In the full rank ($rank = p$) \mathbf{X} case, this is of course, completely parallel to representation (41) and is a consequence of the fact that the columns of *both* \mathbf{U} and \mathbf{Q} (from the QR decomposition of \mathbf{X}) form orthonormal bases for $C(\mathbf{X})$. In general, the two bases are not the same.

¹⁵For a real non-negative definite square matrix (a covariance matrix), the singular value decomposition is the eigen decomposition, $\mathbf{U} = \mathbf{V}$, columns of these matrices are unit eigenvectors, and the SVD singular values are the corresponding eigenvalues.

Now using the singular value decomposition of a full rank (rank p) \mathbf{X} ,

$$\begin{aligned}\mathbf{X}'\mathbf{X} &= \mathbf{V}\mathbf{D}'\mathbf{U}'\mathbf{U}\mathbf{D}\mathbf{V}' \\ &= \mathbf{V}\mathbf{D}^2\mathbf{V}'\end{aligned}\tag{43}$$

which is the eigen (or spectral) decomposition of the symmetric and positive definite $\mathbf{X}'\mathbf{X}$. (The eigenvalues are the squares of the SVD singular values.)

The vector

$$\mathbf{z}_1 \equiv \mathbf{X}\mathbf{v}_1$$

is the product $\mathbf{X}\mathbf{w}$ with the largest squared length in \mathfrak{R}^N subject to the constraint that $\|\mathbf{w}\| = 1$. A second representation of \mathbf{z}_1 is

$$\mathbf{z}_1 = \mathbf{X}\mathbf{v}_1 = \mathbf{U}\mathbf{D}\mathbf{V}'\mathbf{v}_1 = \mathbf{U}\mathbf{D}\begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = d_1\mathbf{u}_1$$

and we see that this largest squared length is d_1^2 and the vector points in the direction of \mathbf{u}_1 . In general,

$$\mathbf{z}_j = \mathbf{X}\mathbf{v}_j = \begin{pmatrix} \langle \mathbf{x}_1, \mathbf{v}_j \rangle \\ \vdots \\ \langle \mathbf{x}_N, \mathbf{v}_j \rangle \end{pmatrix} = d_j\mathbf{u}_j\tag{44}$$

is the vector of the form $\mathbf{X}\mathbf{w}$ with the largest squared length in \mathfrak{R}^N subject to the constraints that $\|\mathbf{w}\| = 1$ and $\langle \mathbf{w}, \mathbf{z}_l \rangle = 0$ for all $l < j$. The squared length is d_j^2 and the vector points in the direction of \mathbf{u}_j .

2.3.1 The Singular Value Decomposition and General Inner Product Spaces

It is potentially useful to consider the relevance of the SVD for matrices to geometry in abstract inner product spaces (e.g. because of the machine learning practice of adopting features that are not elements of a Euclidean space, but rather functions). So, suppose that N vectors $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_N$ span a subspace of the inner product space \mathcal{A} of dimension r and that $\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_r$ form an orthonormal basis for that subspace. Consider then the matrix

$$\mathbf{X}_{N \times r} = \left(\langle \mathbf{w}_i, \mathbf{e}_j \rangle_{\mathcal{A}} \right)_{\substack{i=1,2,\dots,N \\ j=1,2,\dots,r}}\tag{45}$$

\mathbf{X} represents the vectors $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_N$ in the sense that its rows give coefficients to be applied to the elements of the orthonormal basis (the \mathbf{e} s) in order to make linear combinations that are the \mathbf{w} s.

Now, as above, consider the SVD of \mathbf{X} and some related elements of \mathcal{A} . Begin with elements of \mathcal{A} related to the right singular vectors $\mathbf{v}_j \in \mathfrak{R}^r$. Corresponding to them are vectors

$$\mathbf{a}_j = \sum_{l=1}^r v_{jl} \mathbf{e}_l$$

(the real entries of \mathbf{v}_j supplying coefficients for the \mathbf{e} s in order to make up \mathbf{a}_j as a linear combination of the basis vectors). Notice that

$$\langle \mathbf{a}_j, \mathbf{a}_{j'} \rangle_{\mathcal{A}} = \sum_{l=1}^r v_{jl} v_{j'l} = I [j = j']$$

and so $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_r$ form a second orthonormal basis for \mathcal{A} .

Now

$$\begin{aligned} \sum_{i=1}^N \langle \mathbf{w}_i, \mathbf{a}_1 \rangle_{\mathcal{A}}^2 &= \left\| \left(\langle \mathbf{w}_i, \mathbf{a}_1 \rangle_{\mathcal{A}} \right)_{i=1,2,\dots,N} \right\|^2 \\ &= \left\| \left(\sum_{l=1}^r v_{1l} \langle \mathbf{w}_i, \mathbf{e}_l \rangle_{\mathcal{A}} \right)_{i=1,2,\dots,N} \right\|^2 \\ &= \|\mathbf{X} \mathbf{v}_1\|^2 \end{aligned}$$

has the maximum value of

$$\begin{aligned} \sum_{i=1}^N \left\langle \mathbf{w}_i, \sum_{l=1}^r c_l \mathbf{e}_l \right\rangle_{\mathcal{A}}^2 &= \left\| \left(\left\langle \mathbf{w}_i, \sum_{l=1}^r c_l \mathbf{e}_l \right\rangle_{\mathcal{A}} \right)_{i=1,2,\dots,N} \right\|^2 \\ &= \left\| \left(\sum_{l=1}^r c_l \langle \mathbf{w}_i, \mathbf{e}_l \rangle_{\mathcal{A}} \right)_{i=1,2,\dots,N} \right\|^2 \\ &= \|\mathbf{X} \mathbf{c}\|^2 \end{aligned}$$

possible for \mathbf{c} a unit vector in \mathfrak{R}^r and thus $\sum_{l=1}^r c_l \mathbf{e}_l$ a unit vector in \mathcal{A} . That is, \mathbf{a}_1 is a unit vector in \mathcal{A} pointing in a direction such that the projections of the \mathbf{w}_i onto the 1-dimensional subspace of it have the largest possible sum of squared norms. In general, \mathbf{a}_j is a unit vector in \mathcal{A} perpendicular to all of $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_{j-1}$ with maximum sum of squared norms for the projections of the \mathbf{w}_i onto the 1-dimensional subspace of multiples of it.

In a case where a transform T maps \mathfrak{R}^p to an inner product space \mathcal{A} , and one is interested in the subspace of \mathcal{A} of dimension $r \leq N$ spanned by the image of the set of training input vectors

$$\{T(\mathbf{x}_1), T(\mathbf{x}_2), \dots, T(\mathbf{x}_N)\} \quad ,$$

with $\mathbf{w}_i = T(\mathbf{x}_i)$, the foregoing then translates the SVD of the matrix (45) into abstract inner product space geometrical insights concerning transformed training vectors.

2.4 Matrices of Centered Columns and Principal Components

In the event that all the columns of \mathbf{X} have been centered (each $\mathbf{1}'\mathbf{x}_j = 0$ for \mathbf{x}_j the j th column of \mathbf{X}), there is additional terminology and insight associated with singular value decompositions as describing the structure of \mathbf{X} . Note that centering is often sensible in unsupervised learning contexts because the object is to understand the internal structure of the data cases $\mathbf{x}_i \in \mathbb{R}^p$, not the location of the data cloud (that is easily represented by the sample mean vector). So accordingly, we first translate the data cloud to the origin.

Principal components ideas are then based on the singular value decomposition of \mathbf{X}

$$\mathbf{X} = \mathbf{U} \mathbf{D} \mathbf{V}'$$

$N \times p$ $N \times r$ $r \times r$ $r \times p$

(and related spectral/eigen decompositions of $\mathbf{X}'\mathbf{X}$ and $\mathbf{X}\mathbf{X}'$).

2.4.1 "Ordinary" Principal Components

The columns of \mathbf{V} (namely $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_r$) are called the **principal component directions** in \mathbb{R}^p of the \mathbf{x}_i , and the elements of the vectors $\mathbf{z}_j \in \mathbb{R}^N$ from display (44), namely the inner products $\langle \mathbf{x}_i, \mathbf{v}_j \rangle$, are called the **principal components** of the \mathbf{x}_i . (The i th element of \mathbf{z}_j , $\langle \mathbf{x}_i, \mathbf{v}_j \rangle$, is the value of the j th principal component for case i , or the corresponding **principal component score**. The entries of the $p \times 1$ vector \mathbf{v}_j are sometimes called the **component weights** or **loadings** for the j th component. A 0 loading means that the corresponding column of \mathbf{X} is ignored in the creation of \mathbf{z}_j .) Notice that $\langle \mathbf{x}_i, \mathbf{v}_j \rangle \mathbf{v}_j$ is the projection of \mathbf{x}_i onto the 1-dimensional space spanned by \mathbf{v}_j .

Figure 12 provides a summary of the language just introduced. (The $N \times p$ matrix of inner products $\langle \mathbf{x}_i, \mathbf{v}_j \rangle$ is \mathbf{UD} .)

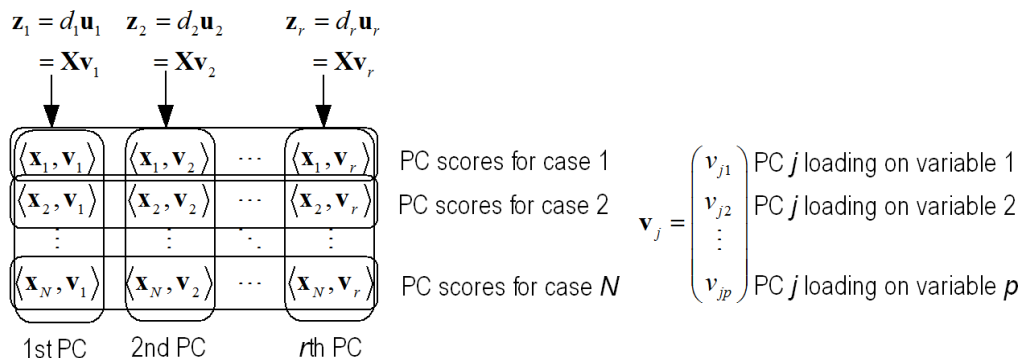


Figure 12: Summary of principal components language.

Figure 13 shows scatterplots of a raw (red) and corresponding standardized (blue) $p = 2$ dataset. The red arrow points in the direction of the *raw data*

first right singular vector (i.e. points "at" the raw data). The blue arrow is in the **first principal component direction** of the *standardized data* (pointing in the direction of their greatest variation).

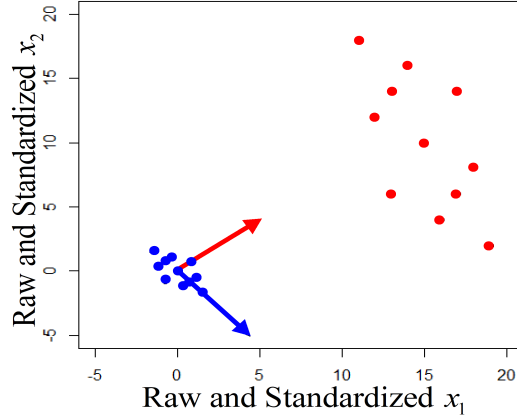


Figure 13: Example of a small $p = 2$ dataset (red dots) and standardized version (blue dots) and (multiples of) the first right singular vector of the dataset and the first principal direction of the standardized dataset.

It is worth thinking a bit more about the form of the product

$$\mathbf{X}^{*l} = \mathbf{U}_l \mathbf{diag}(d_1, d_2, \dots, d_l) \mathbf{V}_l'$$

that we've already said is the best rank l approximation to \mathbf{X} . In fact it is

$$\mathbf{X}^{*l} = \sum_{j=1}^l d_j \mathbf{u}_j \mathbf{v}_j' = \sum_{j=1}^l z_j \mathbf{v}_j' = (z_1, z_2, \dots, z_l) \begin{pmatrix} \mathbf{v}_1' \\ \mathbf{v}_2' \\ \vdots \\ \mathbf{v}_l' \end{pmatrix}$$

and its i th row is $\sum_{j=1}^l \langle \mathbf{x}_i, \mathbf{v}_j \rangle \mathbf{v}_j'$, which (since the \mathbf{v}_j are orthonormal) is the transpose of the projection of \mathbf{x}_i onto $C(\mathbf{V}_l)$. That is,

$$\begin{aligned} \mathbf{X}^{*l} &= \begin{pmatrix} \langle \mathbf{x}_1, \mathbf{v}_1 \rangle \\ \vdots \\ \langle \mathbf{x}_N, \mathbf{v}_1 \rangle \end{pmatrix} \mathbf{v}_1' + \begin{pmatrix} \langle \mathbf{x}_1, \mathbf{v}_2 \rangle \\ \vdots \\ \langle \mathbf{x}_N, \mathbf{v}_2 \rangle \end{pmatrix} \mathbf{v}_2' + \dots + \begin{pmatrix} \langle \mathbf{x}_1, \mathbf{v}_l \rangle \\ \vdots \\ \langle \mathbf{x}_N, \mathbf{v}_l \rangle \end{pmatrix} \mathbf{v}_l' \\ &= z_1 \mathbf{v}_1' + z_2 \mathbf{v}_2' + \dots + z_l \mathbf{v}_l' \\ &= \mathbf{X} \mathbf{v}_1 \mathbf{v}_1' + \mathbf{X} \mathbf{v}_2 \mathbf{v}_2' + \dots + \mathbf{X} \mathbf{v}_l \mathbf{v}_l' \end{aligned}$$

a sum of rank 1 summands, producing for \mathbf{X}^{*l} a matrix with each \mathbf{x}_i in \mathbf{X} replaced by the transpose of its projection onto $C(\mathbf{V}_l)$.

Since $\mathbf{z}_j = d_j \mathbf{u}_j$, $\mathbf{z}_j \mathbf{v}'_j = d_j \mathbf{u}_j \mathbf{v}'_j$. Then since the \mathbf{u}_j s and \mathbf{v}_j s are unit vectors, the sum of squared entries of both \mathbf{z}_j and $\mathbf{z}_j \mathbf{v}'_j$ is d_j^2 . These are non-increasing in j . So the \mathbf{z}_j and $\mathbf{z}_j \mathbf{v}'_j$ decrease in "size" with j , and directions $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_r$ are successively "less important" in describing variation in the \mathbf{x}_i and in reconstructing \mathbf{X} . This agrees with common interpretation of cases where a few singular values are much bigger than the others. There "simple structure" in the data is that observations can be more or less reconstructed as linear combinations of a few orthonormal vectors.

Figure 14 portrays a hypothetical $p = 3$ dataset. Shown are the $N = 9$ data points, the $\text{rank} = 1$ approximation (black balls on the line defined by the first PC direction) and the $\text{rank} = 2$ approximation (black stars on the plane).

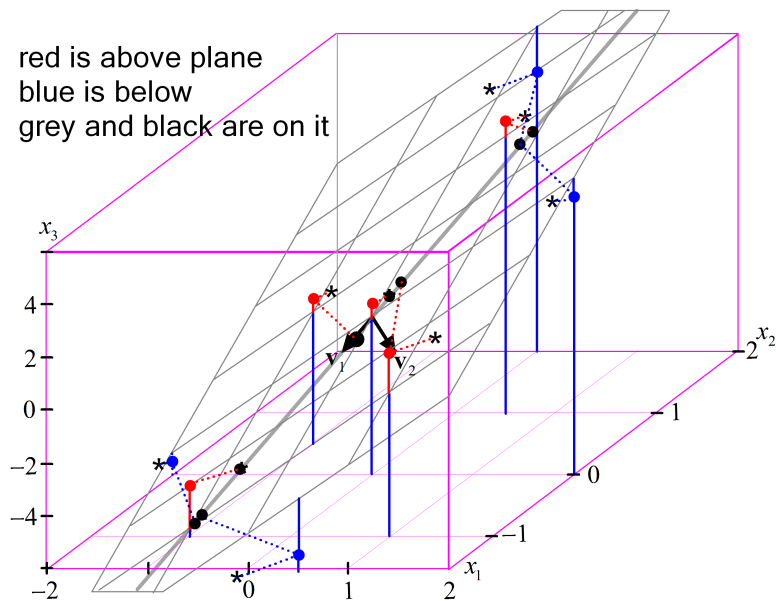


Figure 14: Principal components approximations to a $p = 3$ dataset.

Izenman, in his discussion of "polynomial principal components" points out that in some circumstances the existence of a few very *small* singular values can *also* identify important simple structure in a dataset. Suppose, for example, that all singular values except $d_p \approx 0$ are of appreciable size. One simple feature of the dataset is then that all $\langle \mathbf{x}_i, \mathbf{v}_p \rangle \approx 0$, i.e. there is one linear combination of the p coordinates x_j that is essentially constant (namely $\langle \mathbf{x}, \mathbf{v}_p \rangle$). The data fall nearly on a $(p - 1)$ -dimensional hyperplane in \mathbb{R}^p . In cases where the p coordinates x_j are not functionally independent (for example consisting of centered versions of 1) all values, 2) all squares of values, and 3) all cross products of values of a smaller number of functionally independent variables), a

single "nearly 0" singular value identifies a quadratic function of the functionally independent variables that must be essentially constant, a potentially useful insight about the dataset.

To summarize interpretation of principal components of a centered dataset, one can say the following:

Principal components analysis amounts to the development of an alternative coordinate system in which to represent a p -dimensional dataset. One effectively finds a rotation of the original coordinate system to a new one where axes are defined by the p -vectors $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_r$ in which variation of the data in the directions \mathbf{v}_j decreases with increasing j (as much as possible with each increment of j). The N -vectors \mathbf{u}_j are unit vectors and their multiples $\mathbf{z}_j = d_j \mathbf{u}_j$ are the vectors of coordinates of the N data vectors in the new/rotated coordinate system. (And the d_j are the magnitudes of these vectors of new coordinates in \mathbb{R}^N .)

$\mathbf{X}'\mathbf{X}$ and $\mathbf{X}\mathbf{X}'$ and Principal Components The singular value decomposition of \mathbf{X} means that both $\mathbf{X}'\mathbf{X}$ and $\mathbf{X}\mathbf{X}'$ have useful representations in terms of singular vectors and singular values. Consider first $\mathbf{X}'\mathbf{X}$ (that is most of the sample covariance matrix). As noted in display (43), the SVD of \mathbf{X} means that

$$\mathbf{X}'\mathbf{X} = \mathbf{V}\mathbf{D}^2\mathbf{V}'$$

and it's then clear that the columns of \mathbf{V} are eigenvectors of $\mathbf{X}'\mathbf{X}$ and the squares of the diagonal elements of \mathbf{D} are the corresponding eigenvalues. An eigen analysis of $\mathbf{X}'\mathbf{X}$ then directly yields the principal component directions of the data, and through the further computation of the inner products in (44), the principal components \mathbf{z}_j (and hence the singular vectors \mathbf{u}_j) are available.

Note that

$$\frac{1}{N}\mathbf{X}'\mathbf{X}$$

is the (N -divisor) sample covariance matrix¹⁶ for the p input variables x_1, x_2, \dots, x_p . The principal component directions of \mathbf{X} in \mathbb{R}^p , namely $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_r$, are also unit eigenvectors of the sample covariance matrix. The squared lengths of the principal components \mathbf{z}_j in \mathbb{R}^N divided by N are the (N -divisor) sample variances of entries of the \mathbf{z}_j , and their values are

$$\frac{1}{N}\mathbf{z}'_j\mathbf{z}_j = \frac{1}{N}d_j\mathbf{u}'_j\mathbf{u}_jd_j = \frac{d_j^2}{N}$$

The SVD of \mathbf{X} also implies that

$$\mathbf{X}\mathbf{X}' = \mathbf{U}\mathbf{D}\mathbf{V}'\mathbf{V}\mathbf{D}\mathbf{U}' = \mathbf{U}\mathbf{D}^2\mathbf{U}'$$

¹⁶Notice that **when \mathbf{X} has standardized columns** (i.e. each column of \mathbf{X} , \mathbf{x}_j , has $\langle \mathbf{1}, \mathbf{x}_j \rangle = 0$ and $\langle \mathbf{x}_j, \mathbf{x}_j \rangle = N$), the matrix $\frac{1}{N}\mathbf{X}'\mathbf{X}$ is the sample correlation matrix for the p input variables x_1, x_2, \dots, x_p .

and it's then clear that the columns of \mathbf{U} are eigenvectors of $\mathbf{X}\mathbf{X}'$ and the squares of the diagonal elements of \mathbf{D} are the corresponding eigenvalues. \mathbf{UD} then produces the $N \times r$ matrix of principal components of the data. The principal component directions are unavailable (even indirectly) based only on this second eigen analysis.

2.4.2 "Kernel" Principal Components

Consider first the possibility of using a nonlinear function $\phi : \mathfrak{R}^p \rightarrow \mathfrak{R}^M$ to map data vectors \mathbf{x} to (a usually higher-dimensional) vector of features $\phi(\mathbf{x})$. Of course, this creates a new $N \times M$ data/feature matrix

$$\Phi = \begin{pmatrix} \phi'(\mathbf{x}_1) \\ \phi'(\mathbf{x}_2) \\ \vdots \\ \phi'(\mathbf{x}_N) \end{pmatrix}$$

with entries of Φ belonging to \mathfrak{R} . After centering via

$$\tilde{\Phi} = \Phi - \frac{1}{N}\mathbf{J}\Phi = \left(\mathbf{I} - \frac{1}{N}\mathbf{J}\right)\Phi \quad (46)$$

for \mathbf{J} an $N \times N$ matrix of 1s, one can make a SVD of $\tilde{\Phi}$, producing singular values and both sets of singular vectors for the new feature matrix.

Now, thinking as in Section 1.4.3, suppose \mathcal{K} is a kernel function and one maps data vectors \mathbf{x} to elements $\mathcal{K}(\mathbf{x}, \cdot)$ in the abstract (function) feature space \mathcal{A} . One can think of finding "principal components" for the transformed training set in this feature space. First, the function

$$\bar{\mathcal{K}}(\cdot) \equiv \frac{1}{N} \sum_{i=1}^N \mathcal{K}(\mathbf{x}_i, \cdot)$$

is a well-defined linear combination of the images of the training set in \mathcal{A} and therefore a sensible "center" of the transformed training set. The functions

$$\mathcal{K}(\mathbf{x}_i, \cdot) - \bar{\mathcal{K}}(\cdot) \quad (47)$$

are then sensible centered abstract feature values for the training set. Next, corresponding to the matrix of inner products for a centered set of N points in a Euclidean space is the $N \times N$ matrix of inner products of these centered feature values in the abstract space \mathcal{A} ,

$$\mathbf{C} \equiv \left(\langle \mathcal{K}(\mathbf{x}_i, \cdot) - \bar{\mathcal{K}}(\cdot), \mathcal{K}(\mathbf{x}_j, \cdot) - \bar{\mathcal{K}}(\cdot) \rangle_{\mathcal{A}} \right)_{\substack{i=1, \dots, N \\ j=1, \dots, N}} \quad (48)$$

Then using the basic reproducing kernel fact that $\langle \mathcal{K}(\mathbf{x}, \cdot), \mathcal{K}(\mathbf{z}, \cdot) \rangle_{\mathcal{A}} = \mathcal{K}(\mathbf{x}, \mathbf{z})$ and the notation \mathbf{K} for the Gram matrix (21), it is easy enough to find the representation

$$\mathbf{C} = \mathbf{K} - \frac{1}{N}\mathbf{J}\mathbf{K} - \frac{1}{N}\mathbf{K}\mathbf{J} + \frac{1}{N^2}\mathbf{J}\mathbf{K}\mathbf{J} \quad (49)$$

for the symmetric non-negative definite \mathbf{C} . Finally, an eigen analysis will produce principal components (N vectors of length N of scores) for the training data expressed in the abstract feature space.

To realize the entries in these eigen vectors of kernel principal component scores as inner products of the N functions (47) with "principal component directions" in the abstract feature space, \mathcal{A} , one may return to Section 2.3.1 and begin with any orthonormal basis $\mathcal{E}_1(\cdot), \mathcal{E}_2(\cdot), \dots, \mathcal{E}_N(\cdot)$ for the span of the functions (47) (coming, for example, from use of the Gram-Schmidt process). Then the general inner product space argument beginning with an $N \times N$ matrix with entries $\langle \mathcal{K}(\mathbf{x}_i, \cdot) - \bar{\mathcal{K}}(\cdot), \mathcal{E}_j(\cdot) \rangle_{\mathcal{A}}$ produces N basis functions $\mathcal{V}_1(\cdot), \mathcal{V}_2(\cdot), \dots, \mathcal{V}_N(\cdot)$ whose \mathcal{A} inner products with functions (47) are (up to a sign for each $\mathcal{V}_j(\cdot)$) the entries of the eigen vectors of \mathbf{C} . In cases with small p it may be of interest to examine these abstract principal component direction functions via some plotting.

2.4.3 Graphical (Spectral) Features

Another variant of principal components ideas concerns "graphical spectral features" of a dataset built on thinking of data cases as corresponding to vertices on a graph. This material has emphases in common with the local version of multi-dimensional scaling treated in Section 17.3, and can sometimes provide a way to separate "unconventional" but distinct structures of data points in \mathfrak{R}^p . The basic motivation is to not necessarily look for "convex" groups of points in p -space, but rather for "roughly connected"/"contiguous" sets of points of *any* shape in p -space.

Begin with N vectors $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ in \mathfrak{R}^p . Consider weights $w_{ij} = w(\|\mathbf{x}_i - \mathbf{x}_j\|)$ for a decreasing function $w : [0, \infty) \rightarrow [0, 1]$ and use them to define **similarities/adjacencies** s_{ij} . (For example, we might use $w(d) = \exp(-d^2/c)$ for some $c > 0$.) Similarities can be exactly $s_{ij} = w_{ij}$, but can be even more "locally" defined as follows. For fixed k consider the symmetric set of index pairs

$$\mathcal{N}_k = \left\{ (i, j) \mid \begin{array}{l} \text{the number of } j' \text{ with } w_{ij'} > w_{ij} \text{ is less than } k \\ \text{or the number of } i' \text{ with } w_{i'j} > w_{ij} \text{ is less than } k \end{array} \right\}$$

(an index pair is in the set if one of the items is in the k -nearest neighbor neighborhood of the other). One might then define $s_{ij} = w_{ij} I[(i, j) \in \mathcal{N}_k]$.

In any case, we'll call the matrix

$$\mathbf{S} = (s_{ij})_{\substack{i=1, \dots, N \\ j=1, \dots, N}}$$

the **adjacency matrix**, and use the notation

$$g_i = \sum_{j=1}^N s_{ij}$$

and

$$\mathbf{G} = \mathit{diag}(g_1, g_2, \dots, g_N)$$

It is common to think of the points $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ in \mathfrak{R}^p as nodes/vertices on a graph, with edges between nodes weighted by similarities s_{ij} , and the g_i so-called **node degrees**, i.e. sums of weights of the edges connected to nodes i . In such thinking, $s_{ij} = 0$ indicates that there is no "edge" between case i and case j .

The matrix

$$\mathbf{L} = \mathbf{G} - \mathbf{S}$$

is called the (unnormalized) **graph Laplacian**, and one standardized (with respect to the node degrees) version of this is

$$\tilde{\mathbf{L}} = \mathbf{G}^{-1}\mathbf{L} = \mathbf{I} - \mathbf{G}^{-1}\mathbf{S}$$

and a second standardized version is

$$\mathbf{L}^* = \mathbf{G}^{-1/2}\mathbf{L}\mathbf{G}^{-1/2} = \mathbf{I} - \mathbf{G}^{-1/2}\mathbf{S}\mathbf{G}^{-1/2} \quad (50)$$

Note that for any vector \mathbf{u} ,

$$\begin{aligned} \mathbf{u}'\mathbf{L}\mathbf{u} &= \sum_{i=1}^N g_i u_i^2 - \sum_{i=1}^N \sum_{j=1}^N u_i u_j s_{ij} \\ &= \frac{1}{2} \left(\sum_{i=1}^N \sum_{j=1}^N s_{ij} u_i^2 + \sum_{j=1}^N \sum_{i=1}^N s_{ij} u_j^2 \right) - \sum_{i=1}^N \sum_{j=1}^N u_i u_j s_{ij} \\ &= \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N s_{ij} (u_i - u_j)^2 \end{aligned} \quad (51)$$

so that the $N \times N$ symmetric \mathbf{L} is nonnegative definite. Consider the spectral/eigen decomposition of \mathbf{L} and focus on the *small* eigenvalues. For $\mathbf{v}_1, \dots, \mathbf{v}_m$ eigenvectors corresponding to the 2nd through $(m+1)$ st smallest non-zero eigenvalues (since $\mathbf{L}\mathbf{1} = \mathbf{0}$ there is an uninteresting 0 eigenvalue), let

$$\mathbf{V} = (\mathbf{v}_1, \dots, \mathbf{v}_m)$$

These are "graphical spectral features" and one might think of cases with **similar rows of \mathbf{V}** as "alike." As we noted in the discussion in Section 2.4.1, small eigenvalues are associated with linear combinations of columns of \mathbf{L} that are close to 0.

Why should this work to identify connected structures in a training set? For \mathbf{v}_l a column of \mathbf{V} that is a eigenvector of \mathbf{L} corresponding to a small eigenvalue λ_l , by virtue of relationship (51)

$$\lambda_l = \mathbf{v}_l'\mathbf{L}\mathbf{v}_l = \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N s_{ij} (v_{li} - v_{lj})^2 \approx 0 \quad (52)$$

and points \mathbf{x}_i and \mathbf{x}_j with large adjacencies must have similar corresponding coordinates of the eigenvectors. HTF (at the bottom of their page 545) essentially argue that the number of "0 or nearly 0" eigenvalues of \mathbf{L} is indicative

of the number of connected structures in the original N data vectors. A series of points could be (in sequence) close to successive elements of the sequence but have very small adjacencies for points separated in the sequence. "Structures" by this methodology need NOT be "clumps" of points, but could also be serpentine "chains" of points in \mathfrak{R}^p .

A second version of this is easily built on the **symmetric normalized Laplacian** (50), \mathbf{L}^* . Its eigenvalues are nonnegative and it has a 0 eigenvalue. Let $\lambda_1^* \leq \dots \leq \lambda_m^*$ be the 2nd through $(m+1)$ st smallest eigenvalues and $\mathbf{v}_1^*, \dots, \mathbf{v}_m^*$ be corresponding eigenvectors. Then for λ_l^* such a small non-negative eigenvalue,

$$\lambda_l^* = \mathbf{v}_l^{*'} \mathbf{L}^* \mathbf{v}_l^* = \mathbf{v}_l^{*'} \left(\mathbf{G}^{-1/2} \mathbf{L} \mathbf{G}^{-1/2} \right) \mathbf{v}_l^* = \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \tilde{s}_{ij} \left(\frac{v_{li}^*}{\sqrt{g_i}} - \frac{v_{lj}^*}{\sqrt{g_j}} \right)^2 \approx 0 \quad (53)$$

and points \mathbf{x}_i and \mathbf{x}_j with large adjacencies must have similar corresponding coordinates of the vector $\mathbf{G}^{-1/2} \mathbf{v}_l^*$. So one might treat vectors $\mathbf{G}^{-1/2} \mathbf{v}_l^*$ (or perhaps normalized versions of them) as a second version of m graphical features.

It is also easy to see that

$$\mathbf{P} \equiv \mathbf{G}^{-1} \mathbf{S}$$

is a stochastic matrix and thus specifying an N -state stationary Markov Chain. It is plausible that the standardized graph Laplacian $\tilde{\mathbf{L}} = \mathbf{I} - \mathbf{P}$ identifies groups of states such that transition by such a chain between the groups is relatively infrequent (the MCMC more typically moves within groups).

Part II

Supervised Learning I: Basic Prediction Methodology

3 (Non-OLS) SEL Linear Predictors

There is more to say about the development of a linear predictor

$$\hat{f}(\mathbf{x}) = \mathbf{x}' \hat{\boldsymbol{\beta}}$$

for an appropriate $\hat{\boldsymbol{\beta}} \in \mathfrak{R}^p$ than what is said in books and courses on ordinary linear models (where ordinary least squares is used to fit the linear form to all p input variables or to some subset of M of them). We continue the basic notation of Section 2, where the (supervised learning) problem is prediction, and there is a vector of continuous outputs, \mathbf{Y} , of interest.

3.1 Ridge Regression, the Lasso, and Some Other Shrinking Methods

An alternative to seeking to find a suitable level of complexity in a linear prediction rule through subset selection and least squares fitting of a linear form to the selected variables, is to employ a shrinkage method based on a penalized version of least squares to choose a vector $\hat{\beta} \in \mathfrak{R}^p$ to employ in a linear prediction rule. Here we consider several such methods, all of which have parameters that function as complexity measures and allow $\hat{\beta}$ to range between $\mathbf{0}$ and $\hat{\beta}^{\text{ols}}$ depending upon complexity.

The implementation of these methods is not equivariant to the scaling used to express the input variables x_j . So that we can talk about properties of the methods that are associated with a well-defined scaling, we **assume here that the output variable has been centered** (i.e. that $\langle \mathbf{Y}, \mathbf{1} \rangle = 0$) **and that the columns of \mathbf{X} have been standardized** (and if originally \mathbf{X} had a constant column, it has been removed).

3.1.1 Ridge Regression

For a $\lambda > 0$ the ridge regression coefficient vector $\hat{\beta}_\lambda^{\text{ridge}} \in \mathfrak{R}^p$ is

$$\hat{\beta}_\lambda^{\text{ridge}} = \arg \min_{\beta \in \mathfrak{R}^p} \{ (\mathbf{Y} - \mathbf{X}\beta)' (\mathbf{Y} - \mathbf{X}\beta) + \lambda \beta' \beta \} \quad (54)$$

Here λ is a penalty/complexity parameter that controls how much $\hat{\beta}^{\text{ols}}$ is shrunken towards $\mathbf{0}$. The unconstrained minimization problem expressed in (54) has an equivalent constrained minimization description as

$$\hat{\beta}_t^{\text{ridge}} = \arg \min_{\beta \text{ with } \|\beta\|^2 \leq t} (\mathbf{Y} - \mathbf{X}\beta)' (\mathbf{Y} - \mathbf{X}\beta) \quad (55)$$

for an appropriate $t > 0$. (Corresponding to λ used in form (54), is $t = \|\hat{\beta}_\lambda^{\text{ridge}}\|^2$ used in display (55). Conversely, corresponding to t used in form (55), one may use a value of λ in display (54) producing the same error sum of squares.) Figure 15 is a representation of the constrained version of the ridge optimization problem for $p = 2$. Pictured are a contour plot for the quadratic error sum of squares $(\mathbf{Y} - \mathbf{X}\beta)' (\mathbf{Y} - \mathbf{X}\beta)$ function of β , the constraint region for β , $\hat{\beta}^{\text{ols}}$ and $\hat{\beta}_t^{\text{ridge}}$.

The unconstrained form (54) calls upon one to minimize

$$(\mathbf{Y} - \mathbf{X}\beta)' (\mathbf{Y} - \mathbf{X}\beta) + \lambda \beta' \beta$$

and some vector calculus leads directly to

$$\hat{\beta}_\lambda^{\text{ridge}} = (\mathbf{X}'\mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}'\mathbf{Y}$$

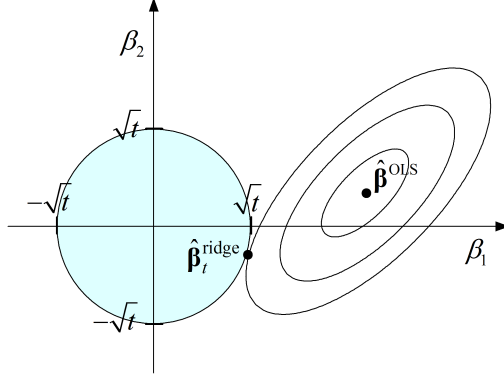


Figure 15: Cartoon Representing the Constrained Version of Ridge Optimization for $p = 2$

So then, using the singular value decomposition of \mathbf{X} (with $rank = r$),

$$\begin{aligned}
\widehat{\mathbf{Y}}_{\lambda}^{\text{ridge}} &= \mathbf{X} \widehat{\boldsymbol{\beta}}_{\lambda}^{\text{ridge}} \\
&= \mathbf{U} \mathbf{D} \mathbf{V}' (\mathbf{V} \mathbf{D} \mathbf{U}' \mathbf{U} \mathbf{D} \mathbf{V}' + \lambda \mathbf{I})^{-1} \mathbf{V} \mathbf{D} \mathbf{U}' \mathbf{Y} \\
&= \mathbf{U} \mathbf{D} (\mathbf{V}' (\mathbf{V} \mathbf{D} \mathbf{U}' \mathbf{U} \mathbf{D} \mathbf{V}' + \lambda \mathbf{I}) \mathbf{V})^{-1} \mathbf{D} \mathbf{U}' \mathbf{Y} \\
&= \mathbf{U} \mathbf{D} (\mathbf{D}^2 + \lambda \mathbf{I})^{-1} \mathbf{D} \mathbf{U}' \mathbf{Y} \\
&= \sum_{j=1}^r \left(\frac{d_j^2}{d_j^2 + \lambda} \right) \langle \mathbf{Y}, \mathbf{u}_j \rangle \mathbf{u}_j
\end{aligned} \tag{56}$$

Comparing to equation (42) and recognizing that

$$0 < \frac{d_{j+1}^2}{d_{j+1}^2 + \lambda} \leq \frac{d_j^2}{d_j^2 + \lambda} < 1$$

we see that the coefficients of the orthonormal basis vectors \mathbf{u}_j employed to get $\widehat{\mathbf{Y}}_{\lambda}^{\text{ridge}}$ are shrunken version of the coefficients applied to get $\widehat{\mathbf{Y}}^{\text{ols}}$. The most severe shrinking is enforced in the directions of the smallest principal components of \mathbf{X} (the \mathbf{u}_j least important in making up low rank approximations to \mathbf{X}). Since from representation (56)

$$\left\| \widehat{\mathbf{Y}}_{\lambda}^{\text{ridge}} \right\|^2 = \sum_{j=1}^r \left(\frac{d_j^2}{d_j^2 + \lambda} \right)^2 \langle \mathbf{Y}, \mathbf{u}_j \rangle^2$$

the "size" of the ridge prediction vector for the N centered responses is decreasing in λ .

Notice also from representation (56) that

$$\begin{aligned}\widehat{\mathbf{Y}}_\lambda^{\text{ridge}} &= \sum_{j=1}^r \left(\frac{1}{d_j^2 + \lambda} \right) \langle \mathbf{Y}, \mathbf{X} \mathbf{v}_j \rangle \mathbf{X} \mathbf{v}_j \\ &= \mathbf{X} \sum_{j=1}^r \left(\frac{1}{d_j^2 + \lambda} \right) \langle \mathbf{Y}, \mathbf{X} \mathbf{v}_j \rangle \mathbf{v}_j\end{aligned}$$

so that

$$\widehat{\boldsymbol{\beta}}_\lambda^{\text{ridge}} = \sum_{j=1}^r \left(\frac{1}{d_j^2 + \lambda} \right) \langle \mathbf{Y}, \mathbf{X} \mathbf{v}_j \rangle \mathbf{v}_j$$

and

$$\|\widehat{\boldsymbol{\beta}}_\lambda^{\text{ridge}}\|^2 = \sum_{j=1}^r \left(\frac{1}{d_j^2 + \lambda} \right)^2 \langle \mathbf{Y}, \mathbf{X} \mathbf{v}_j \rangle^2$$

which is also clearly decreasing in λ . An upshot of these facts about "shrinking" is that one can think of (the penalty parameter) λ as a complexity parameter that defines paths in \mathfrak{R}^N and \mathfrak{R}^p from OLS predictions and coefficients to degenerate ($\mathbf{0}$) ones, passing through a spectrum of plausible (ridge) linear predictors.

There is an interesting "grouping effect" associated with ridge regression. This is that highly correlated inputs, say x_j and $x_{j'}$, (being already standardized so they have sample standard deviation 1 across the training set) will have ridge regression coefficients of essentially the same magnitude. This can be understood as follows. Without loss of generality, assume that x_j and $x_{j'}$ are highly positively correlated (so that they are essentially the same variable). For any regression coefficients β_j and $\beta_{j'}$ and number α (including $\beta_j / (\beta_j + \beta_{j'})$) the contribution of x_j and $x_{j'}$ to \hat{y} (and thus the error sum of squares) is

$$\beta_j x_j + \beta_{j'} x_{j'} \approx \alpha (\beta_j + \beta_{j'}) x_j + (1 - \alpha) (\beta_j + \beta_{j'}) x_{j'}$$

But the contribution of $\alpha (\beta_j + \beta_{j'})$ and $(1 - \alpha) (\beta_j + \beta_{j'})$ to the sum of squared regression coefficients is

$$\alpha^2 (\beta_j + \beta_{j'})^2 + (1 - \alpha)^2 (\beta_j + \beta_{j'})^2 = (\alpha^2 + (1 - \alpha)^2) (\beta_j + \beta_{j'})^2$$

which is minimum at $\alpha = 1/2$, where the coefficients for x_j and $x_{j'}$ are the same.

The function

$$\begin{aligned}
df(\lambda) &= \text{tr} \left(\mathbf{X} (\mathbf{X}'\mathbf{X} + \lambda\mathbf{I})^{-1} \mathbf{X}' \right) \\
&= \text{tr} \left(\mathbf{U}\mathbf{D} (\mathbf{D}^2 + \lambda\mathbf{I})^{-1} \mathbf{D}\mathbf{U}' \right) \\
&= \text{tr} \left(\sum_{j=1}^r \left(\frac{d_j^2}{d_j^2 + \lambda} \right) \mathbf{u}_j \mathbf{u}_j' \right) \\
&= \text{tr} \left(\sum_{j=1}^r \left(\frac{d_j^2}{d_j^2 + \lambda} \right) \mathbf{u}_j' \mathbf{u}_j \right) \\
&= \sum_{j=1}^r \left(\frac{d_j^2}{d_j^2 + \lambda} \right)
\end{aligned}$$

is called the "effective degrees of freedom" associated with the ridge regression. In regard to this choice of nomenclature, note that if $\lambda = 0$ ridge regression is ordinary least squares and this is r , the usual degrees of freedom associated with projection onto $C(\mathbf{X})$, i.e. trace of the projection matrix onto this column space.

As $\lambda \rightarrow \infty$, the effective degrees of freedom goes to 0 as (the centered) $\widehat{\mathbf{Y}}_\lambda^{\text{ridge}}$ goes to $\mathbf{0}$ (corresponding to a constant predictor). Notice also (for future reference) that since $\widehat{\mathbf{Y}}_\lambda^{\text{ridge}} = \mathbf{X}\widehat{\boldsymbol{\beta}}_\lambda^{\text{ridge}} = \mathbf{X}(\mathbf{X}'\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}'\mathbf{Y} = \mathbf{M}\mathbf{Y}$ for $\mathbf{M} = \mathbf{X}(\mathbf{X}'\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}'$, if one assumes that

$$\text{Cov}\mathbf{Y} = \sigma^2\mathbf{I}$$

(conditioned on the \mathbf{x}_i in the training data, the outputs are uncorrelated and have constant variance σ^2) then

$$\text{effective degrees of freedom} = \text{tr}(\mathbf{M}) = \frac{1}{\sigma^2} \sum_{i=1}^N \text{Cov}(\hat{y}_i, y_i) \quad (57)$$

This follows since $\widehat{\mathbf{Y}} = \mathbf{M}\mathbf{Y}$ and $\text{Cov}\mathbf{Y} = \sigma^2\mathbf{I}$ imply that

$$\begin{aligned}
\text{Cov} \begin{pmatrix} \widehat{\mathbf{Y}} \\ \mathbf{Y} \end{pmatrix} &= \sigma^2 \begin{pmatrix} \mathbf{M} \\ \mathbf{I} \end{pmatrix} \mathbf{I} (\mathbf{M}'|\mathbf{I}) \\
&= \sigma^2 \begin{pmatrix} \mathbf{M}\mathbf{M}' & \mathbf{M} \\ \mathbf{M}' & \mathbf{I} \end{pmatrix}
\end{aligned}$$

and the terms $\text{Cov}(\hat{y}_i, y_i)$ are the diagonal elements of the upper right block of this covariance matrix. This suggests that $\text{tr}(\mathbf{M})$ is a plausible general definition for effective degrees of freedom for any *linear* fitting method $\widehat{\mathbf{Y}} = \mathbf{M}\mathbf{Y}$, and that *more generally*, the last form in form (57) might be used in

situations where $\widehat{\mathbf{Y}}$ is other than a linear form in \mathbf{Y} . Further (reasonably enough) the last form is a measure of how strongly the outputs in the training set can be expected to be related to their predictions.

Further, in the linear case with $\widehat{\mathbf{Y}} = \mathbf{M} \mathbf{Y}$,

$$\text{effective degrees of freedom} = \text{tr}(\mathbf{M}) = \sum_{i=1}^N \frac{\partial \hat{y}_i}{\partial y_i}$$

and we see that the effective degrees of freedom is some total measure of how sensitive predictions are at the training inputs \mathbf{x}_i to the corresponding training values y_i . This raises at least the possibility that in nonlinear cases, an approximate/estimated value of the general effective degrees of freedom (57) might be the random variable

$$\sum_{i=1}^N \frac{\partial \hat{y}_i}{\partial y_i} \Big|_{\mathbf{Y}}$$

3.1.2 The Lasso, Etc.

The "lasso" (least absolute selection and shrinkage operator) and some other relatives of ridge regression are the result of generalizing the optimization criteria (54) and (55) by replacing $\beta' \beta = \|\beta\|^2 = \sum_{j=1}^p \beta_j^2$ with $\sum_{j=1}^p |\beta_j|^q$ for a $q > 0$. That produces

$$\widehat{\beta}_\lambda^q = \arg \min_{\beta \in \mathfrak{R}^p} \left\{ (\mathbf{Y} - \mathbf{X}\beta)' (\mathbf{Y} - \mathbf{X}\beta) + \lambda \sum_{j=1}^p |\beta_j|^q \right\} \quad (58)$$

generalizing form (54) and

$$\widehat{\beta}_t^q = \arg \min_{\beta \text{ with } \sum_{j=1}^p |\beta_j|^q \leq t} (\mathbf{Y} - \mathbf{X}\beta)' (\mathbf{Y} - \mathbf{X}\beta) \quad (59)$$

generalizing form (55). The so called "lasso" is the $q = 1$ case of form (58) and form (59) and in general, these have been called the "**bridge regression**" problems. That is, for $t > 0$

$$\widehat{\beta}_t^{\text{lasso}} = \arg \min_{\beta \text{ with } \sum_{j=1}^p |\beta_j| \leq t} (\mathbf{Y} - \mathbf{X}\beta)' (\mathbf{Y} - \mathbf{X}\beta) \quad (60)$$

Because of the shape of the constraint region

$$\left\{ \beta \in \mathfrak{R}^p \mid \sum_{j=1}^p |\beta_j| \leq t \right\}$$

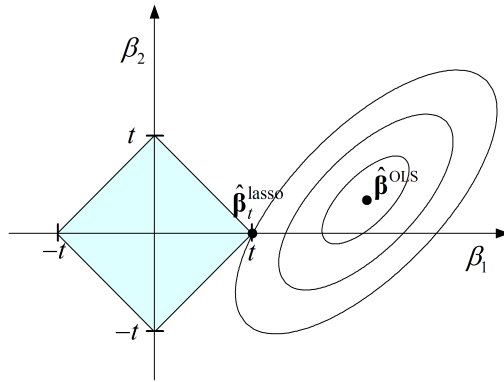


Figure 16: Cartoon Representing the Constrained Version of Lasso Optimization for $p = 2$

(in particular its sharp corners at coordinate axes) some coordinates of $\hat{\beta}_t^{\text{lasso}}$ are often 0, and the lasso automatically provides simultaneous shrinking of $\hat{\beta}^{\text{ols}}$ toward $\mathbf{0}$ and rational subset selection. (The same is true of cases of form (59) with $q < 1$.)

Figure 16 is a representation of the constrained version of the lasso optimization problem for $p = 2$. Pictured are a contour plot for the quadratic error sum of squares $(\mathbf{Y} - \mathbf{X}\beta)'(\mathbf{Y} - \mathbf{X}\beta)$ function of β , the constraint region for β , $\hat{\beta}^{\text{ols}}$ and $\hat{\beta}_t^{\text{lasso}}$.

For comparison purposes, Figure 17 provides representations of $p = 2$ bridge regression constraint regions for $t = 1$. For $q < 1$ the regions not only have "corners," but are not convex.

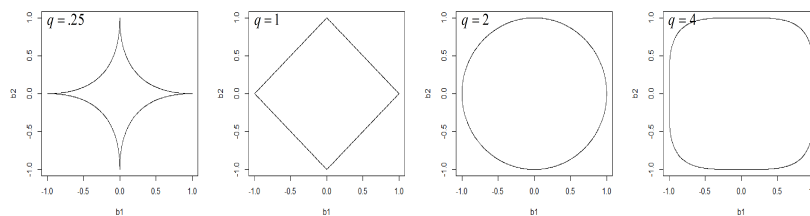


Figure 17: $p = 2$ "bridge" constraint regions for $t = 1$.

It is not obvious how to produce a useful version of formula (57), i.e.

$$\text{effective degrees of freedom} = \frac{1}{\sigma^2} \sum_{i=1}^N \text{Cov}(\hat{y}_i, y_i)$$

for the lasso. But Zhou, Hastie, and Tibshirani in 2007 (*AOS*) argued that

this is the mean number of non-zero components of $\widehat{\beta}_\lambda^{\text{lasso}}$. Obviously then, the random variable

$$\widehat{\text{df}}(\lambda) = \text{the number of non-zero components of } \widehat{\beta}_\lambda^{\text{lasso}}$$

is an unbiased estimator of the effective degrees of freedom.

There are a number of modifications of the ridge/lasso idea. One is the "elastic net" idea. This is a compromise between the ridge and lasso methods. For an $\alpha \in (0, 1)$ and some $t > 0$, this is defined by

$$\widehat{\beta}_{\alpha,t}^{\text{enet}} = \underset{\beta \text{ with } \sum_{j=1}^p ((1-\alpha)|\beta_j| + \alpha\beta_j^2) \leq t}{\text{arg min}} (\mathbf{Y} - \mathbf{X}\beta)' (\mathbf{Y} - \mathbf{X}\beta)$$

(The constraint is a compromise between the ridge and lasso constraints.) For comparison purposes, Figure 18 provides some representations of $p = 2$ elastic net constraint regions for $t = 3$ (made using some code of Prof. Huaqing Wu) that clearly show the compromise nature of the elastic net. The constraint regions have "corners" like the lasso regions but are otherwise more rounded than the lasso regions.

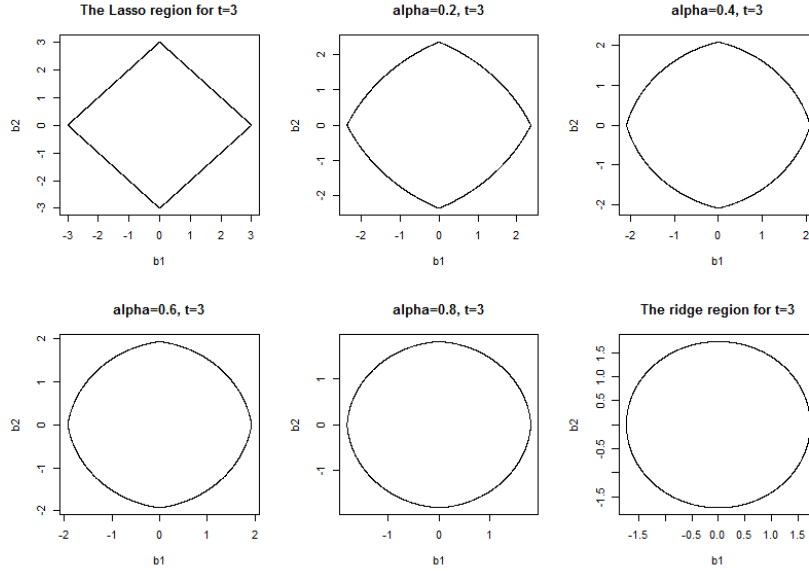


Figure 18: Some $p = 2$ elastic net constraint regions for $t = 3$.

The equivalent unconstrained optimization specification of elastic net fitted coefficient vectors is for $\lambda_1 > 0$ and $\lambda_2 > 0$

$$\widehat{\beta}_{\lambda_1, \lambda_2}^{\text{enet}} = \underset{\beta \in \mathbb{R}^p}{\text{arg min}} \left\{ (\mathbf{Y} - \mathbf{X}\beta)' (\mathbf{Y} - \mathbf{X}\beta) + \lambda_1 \sum_{j=1}^p |\beta_j| + \lambda_2 \sum_{j=1}^p \beta_j^2 \right\}$$

Several sources (including a 2005 *JRSSB* paper of Zhou and Hastie) suggest that a modification of the elastic net idea, namely

$$(1 + \lambda_2) \widehat{\boldsymbol{\beta}}_{\lambda_1, \lambda_2}^{\text{enet}} \quad (61)$$

performs better than the original version.

For $\widehat{\boldsymbol{\beta}}_{\lambda_1, \lambda_2}^{\text{enet}}$ with r non-zero components and \mathbf{X}_* made up of the corresponding columns of \mathbf{X} , estimated effective degrees of freedom for the unmodified form of the elastic net are

$$\text{df}(\widehat{\lambda}_1, \widehat{\lambda}_2) = \text{tr} \left(\mathbf{X}_* (\mathbf{X}_* \mathbf{X}_* + \lambda_2 \mathbf{I})^{-1} \mathbf{X}_*' \right) = \sum_{j=1}^r \left(\frac{d_j^2}{d_j^2 + \lambda_2} \right) \quad (62)$$

(for d_j s the singular values of \mathbf{X}_*). The modified form (61) has estimated effective degrees of freedom $(1 + \lambda_2)$ times this value (62).

Breiman proposed a different shrinkage methodology he called the **nonnegative garotte** that attempts to find "optimal" reweightings of the elements of $\widehat{\boldsymbol{\beta}}^{\text{ols}}$. That is, for $\lambda > 0$ Breiman considered the vector optimization problem defined by

$$c_\lambda = \underset{\mathbf{c} \in \mathbb{R}^p \text{ with } c_j \geq 0, j=1, \dots, p}{\text{arg min}} \left\{ \left(\mathbf{Y} - \mathbf{X} \text{diag}(\mathbf{c}) \widehat{\boldsymbol{\beta}}^{\text{ols}} \right)' \left(\mathbf{Y} - \mathbf{X} \text{diag}(\mathbf{c}) \widehat{\boldsymbol{\beta}}^{\text{ols}} \right) + \lambda \sum_{j=1}^p c_j \right\}$$

and the corresponding fitted coefficient vector

$$\widehat{\boldsymbol{\beta}}_\lambda^{\text{ngg}} = \text{diag}(c_\lambda) \widehat{\boldsymbol{\beta}}^{\text{ols}} = \begin{pmatrix} c_{\lambda 1} \widehat{\beta}_1^{\text{ols}} \\ \vdots \\ c_{\lambda p} \widehat{\beta}_p^{\text{ols}} \end{pmatrix}$$

HTF provide explicit formulas for fitted coefficients for the special case of \mathbf{X} with **orthonormal** columns. (The table below is mostly their Table 4.3.)

Method of Fitting	Fitted Coefficient for x_j
OLS	$\widehat{\beta}_j^{\text{ols}}$
Best Subset (of Size M)	$\widehat{\beta}_j^{\text{ols}} I \left[\text{rank} \left \widehat{\beta}_j^{\text{ols}} \right \leq M \right]$
Ridge Regression	$\widehat{\beta}_j^{\text{ols}} \left(\frac{1}{1 + \lambda} \right)$
Lasso and $(1 + \lambda_2) \widehat{\boldsymbol{\beta}}_{\lambda_1, \lambda_2}^{\text{enet}}$	$\left(\text{sign} \widehat{\beta}_j^{\text{ols}} \right) \left(\left \widehat{\beta}_j^{\text{ols}} \right - \frac{\lambda}{2} \right)_+$
Elastic Net	$\frac{1}{1 + \lambda_2} \left(\text{sign} \widehat{\beta}_j^{\text{ols}} \right) \left(\left \widehat{\beta}_j^{\text{ols}} \right - \frac{\lambda_1}{2} \right)_+$
Nonnegative Garotte	$\widehat{\beta}_j^{\text{ols}} \left(1 - \frac{\lambda}{2 \left(\widehat{\beta}_j^{\text{ols}} \right)^2} \right)_+$

These formulas show that best subset regression provides a kind of "hard thresholding" of the least squares coefficients (setting all but the M largest to 0) and ridge regression provides (the same) shrinking of all coefficients toward 0. Both the lasso and the nonnegative garotte provide a kind of "soft thresholding" of the coefficients (typically "zeroing out" some small ones). The elastic net provides both the ridge type shrinkage of all the coefficients and the lasso soft thresholding. Note that in this "orthonormal columns" case, modification of the elastic net coefficient vector as in formula (61) simply reduces it to a corresponding lasso coefficient vector. (When the predictors are not orthogonal, i.e. uncorrelated, one can expect the modified elastic net to be something other than a lasso.)

For comparison purposes, Figure 19 provides plots of the functions (in the previous table) of OLS coefficients giving ridge (blue), lasso (red), and nonnegative garotte (green) coefficients for the "orthonormal predictors" case. (Solid lines are $\lambda = 1$ plots and dotted ones are for $\lambda = 3$.)

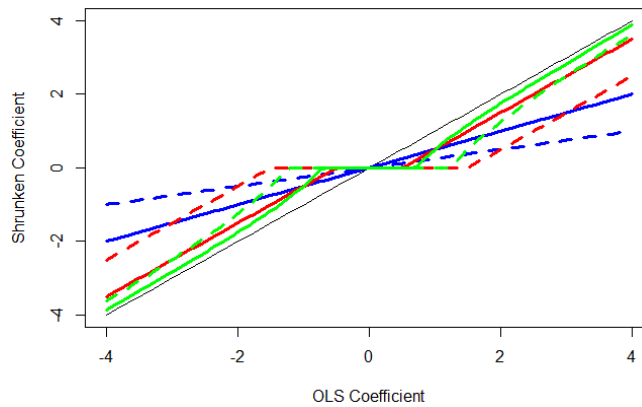


Figure 19: Plots of shrunken coefficients for the "orthonormal inputs x_j " case. Ridge is (blue), lasso is (red), and nonnegative garotte is (green).

By now, a wide variety of lasso-like penalized least squares methods have been suggested, tailored to a variety of special circumstances (and are discussed, for example, by Hastie, Tibshirani and Wainwright). Notable are so-called "group lasso," "sparse group lasso," and "fused lasso" methods. To give the flavor of what has been proposed, we'll illustrate the (2-) group lasso. If for some reason the coordinates of $\mathbf{x} \in \mathbb{R}^p$ break naturally into 2 groups (say the first l and last $p - l$ coordinates of \mathbf{x}). For a $\lambda > 0$, a "group lasso" coefficient vector is of the form

$$\hat{\beta}_\lambda^{\text{group lasso}} = \arg \min_{\beta \in \mathbb{R}^p} \left\{ (\mathbf{Y} - \mathbf{X}\beta)' (\mathbf{Y} - \mathbf{X}\beta) + \lambda \left(\sqrt{\sum_{j=1}^l \beta_j^2} + \sqrt{\sum_{j=l+1}^p \beta_j^2} \right) \right\}$$

Of course, there can be more than 2 groups, and in the event that each group

is of size 1 this reduces to the simple lasso.

Looking at the geometry of the kind of constraint regions that are associated with this methodology, it's plausible (and correct) that it tends to "zero-out" coefficients in groups associated with the penalty. Figure 20 provides a representation of a $p = 3$ constraint region associated with a grouped lasso where coordinates 1 and 2 of \mathbf{x} are grouped separate from coordinate 3. The corresponding lasso region is shown for comparison purposes.

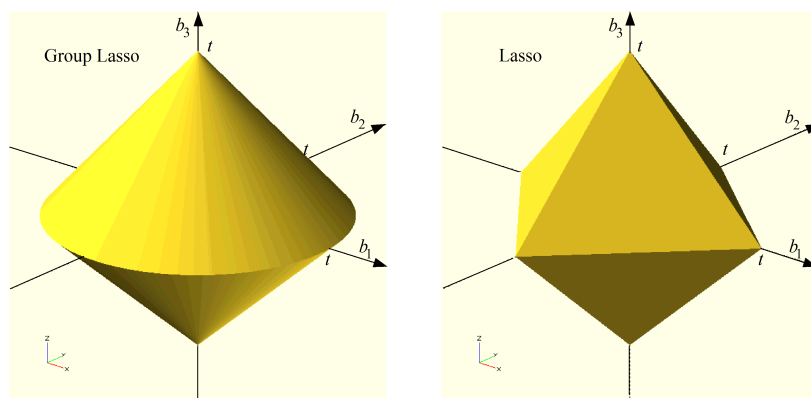


Figure 20: A $p = 3$ constraint region associated with a grouped lasso where coordinates 1 and 2 of \mathbf{x} are grouped separate from coordinate 3.

The development of the lasso and related predictors has been built on minimization of a penalized version of the error sum of squares, $N\overline{\text{EF}}$ for SEL. All of the theory and representations here are special to this case. But as a practical matter, as long as one has an effective/appropriate optimization algorithm there is nothing to prevent consideration of other losses. Possibilities include at least

1. using a negative Bernoulli loglikelihood as a loss and considering penalized logistic regression (either as simply a means of fitting $P[y = 1|\mathbf{x}]$, or for purposes of producing a good voting function for classification), or
2. using a penalized exponential or hinge loss as in Section 1.5.3 for purposes of producing a good voting function for classification, or
3. using a penalized negative AUC loss for producing a good ordering function \mathcal{O} .

The first of these is an option in the famous `glmnet` package in R.

3.1.3 Least Angle Regression (LAR)

Another class of shrinkage methods is defined algorithmically, rather than directly algebraically, or in terms of solutions to optimization problems. This includes the LAR (least angle regression). A description of the whole set of

LAR regression parameters $\hat{\beta}^{\text{LAR}}$ (for the case of \mathbf{X} with each column centered and with norm 1 and centered \mathbf{Y}) follows. (This is some kind of amalgam of the descriptions of Izenman, CFZ, and the presentation in the 2003 paper *Least Angle Regression* by Efron, Hastie, Johnstone, and Tibshirani.)

Note that for $\hat{\mathbf{Y}}$ a vector of predictions, the vector

$$\hat{\mathbf{c}} \equiv \mathbf{X}' (\mathbf{Y} - \hat{\mathbf{Y}})$$

has elements that are proportional to the correlations between the columns of \mathbf{X} (the \mathbf{x}_j) and the residual vector $\mathbf{R} = \mathbf{Y} - \hat{\mathbf{Y}}$. We'll let

$$\hat{C} = \max_j |\hat{c}_j| \quad \text{and} \quad s_j = \text{sign}(\hat{c}_j)$$

Notice also that if \mathbf{X}_l is some matrix made up of l linearly independent columns of \mathbf{X} , then if $\hat{\beta}$ is an l -vector and $\mathbf{W} = \mathbf{X}_l \hat{\beta}$, $\hat{\beta}$ can be recovered from \mathbf{W} as $(\mathbf{X}_l' \mathbf{X}_l)^{-1} \mathbf{X}_l' \mathbf{W}$. (This latter means that if we define a path for $\hat{\mathbf{Y}}$ vectors in $C(\mathbf{X})$ and know for each $\hat{\mathbf{Y}}$ which linearly independent set of columns of \mathbf{X} is used in its creation, we can recover the corresponding path $\hat{\beta}$ takes through \mathbb{R}^p .)

1. Begin with $\hat{\mathbf{Y}}_0 = \mathbf{0}$, $\hat{\beta}_0 = \mathbf{0}$, and $\mathbf{R}_0 = \mathbf{Y} - \hat{\mathbf{Y}} = \mathbf{Y}$ and find

$$j_1 = \arg \max_j |\langle \mathbf{x}_j, \mathbf{Y} \rangle|$$

(the index of the predictor x_j most strongly correlated with y) and add j_1 to an (initially empty) "active set" of indices, A .

2. Move $\hat{\mathbf{Y}}$ from $\hat{\mathbf{Y}}_0$ in the direction of the projection of \mathbf{Y} onto the space spanned by \mathbf{x}_{j_1} (namely $\langle \mathbf{x}_{j_1}, \mathbf{Y} \rangle \mathbf{x}_{j_1}$) until there is another index $j_2 \neq j_1$ with

$$|\hat{c}_{j_2}| = \left| \langle \mathbf{x}_{j_2}, \mathbf{Y} - \hat{\mathbf{Y}} \rangle \right| = \left| \langle \mathbf{x}_{j_1}, \mathbf{Y} - \hat{\mathbf{Y}} \rangle \right| = |\hat{c}_{j_1}|$$

At that point, call the current vector of predictions $\hat{\mathbf{Y}}_1$ and the corresponding current parameter vector $\hat{\beta}_1$ and add index j_2 to the active set A . As it turns out, for

$$\gamma_1 = \min_{j \neq j_1} \left\{ \left(\frac{\hat{C}_0 - \hat{c}_{0j}}{1 - \langle \mathbf{x}_j, \mathbf{x}_{j_1} \rangle} \right), \left(\frac{\hat{C}_0 + \hat{c}_{0j}}{1 + \langle \mathbf{x}_j, \mathbf{x}_{j_1} \rangle} \right) \right\}^+$$

(where the "+" indicates that only positive values are included in the minimization) $\hat{\mathbf{Y}}_1 = \hat{\mathbf{Y}}_0 + s_{j_1} \gamma_1 \mathbf{x}_{j_1} = s_{j_1} \gamma_1 \mathbf{x}_{j_1}$ and $\hat{\beta}_1$ is a vector of all 0s except for $s_{j_1} \gamma_1$ in the j_1 position. Let $\mathbf{R}_1 = \mathbf{Y} - \hat{\mathbf{Y}}_1$.

3. At stage l with A of size l , $\hat{\beta}_{l-1}$ (with only $l-1$ non-zero entries), $\hat{\mathbf{Y}}_{l-1}$, $\mathbf{R}_{l-1} = \mathbf{Y} - \hat{\mathbf{Y}}_{l-1}$, and $\hat{\mathbf{c}}_{l-1} = \mathbf{X}' \mathbf{R}_{l-1}$ in hand, move from $\hat{\mathbf{Y}}_{l-1}$ toward the projection of \mathbf{Y} onto the sub-space of \mathbb{R}^N spanned by $\{\mathbf{x}_{j_1}, \dots, \mathbf{x}_{j_l}\}$.

This is (as it turns out) in the direction of a unit \mathbf{u}_l vector "making equal angles less than 90 degrees with all \mathbf{x}_j with $j \in A$ " until there is an index $j_{l+1} \notin A$ with

$$|\hat{c}_{l-1,j+1}| = |\hat{c}_{l-1,j_1}| \quad (= |\hat{c}_{l-1,j_2}| = \dots = |\hat{c}_{l-1,j_l}|)$$

At that point, with $\hat{\mathbf{Y}}_l$ the current vector of predictions, let $\hat{\boldsymbol{\beta}}_l$ (with only l non-zero entries) be the corresponding coefficient vector, take $\mathbf{R}_l = \mathbf{Y} - \hat{\mathbf{Y}}_l$ and $\hat{\mathbf{c}}_l = \mathbf{X}'\mathbf{R}_l$. It can be argued that with

$$\gamma_l = \min_{j \notin A} \left\{ \left(\frac{\hat{C}_{l-1} - \hat{c}_{l-1,j}}{1 - \langle \mathbf{x}_j, \mathbf{u}_l \rangle} \right), \left(\frac{\hat{C}_{l-1} + \hat{c}_{l-1,j}}{1 + \langle \mathbf{x}_j, \mathbf{u}_l \rangle} \right) \right\}^+$$

$\hat{\mathbf{Y}}_l = \hat{\mathbf{Y}}_{l-1} + s_{j_{l+1}}\gamma_l\mathbf{u}_l$. Add the index j_{l+1} to the set of active indices, A , and repeat.

This continues until there are $r = \text{rank}(\mathbf{X})$ indices in A , and at that point $\hat{\mathbf{Y}}$ moves from $\hat{\mathbf{Y}}_{r-1}$ to $\hat{\mathbf{Y}}^{\text{ols}}$ and $\hat{\boldsymbol{\beta}}$ moves from $\hat{\boldsymbol{\beta}}_{r-1}$ to $\hat{\boldsymbol{\beta}}^{\text{ols}}$ (the version of an OLS coefficient vector with non-zero elements only in positions with indices in A). This defines a piecewise linear path for $\hat{\mathbf{Y}}$ (and therefore $\hat{\boldsymbol{\beta}}$) that could, for example, be parameterized by $\|\hat{\mathbf{Y}}\|$ or $\|\mathbf{Y} - \hat{\mathbf{Y}}\|$.

There are several issues raised by the description above. For one, the standard exposition of this method seems to be that the direction vector \mathbf{u}_l is prescribed by letting $\mathbf{W}_l = (s_{j_1}\mathbf{x}_{j_1}, \dots, s_{j_l}\mathbf{x}_{j_l})$ and taking

$$\mathbf{u}_l = \frac{1}{\|\mathbf{W}_l(\mathbf{W}_l'\mathbf{W}_l)^{-1}\mathbf{1}\|} \mathbf{W}_l(\mathbf{W}_l'\mathbf{W}_l)^{-1}\mathbf{1}$$

It's clear that $\mathbf{W}_l'\mathbf{u}_l = \|\mathbf{W}_l(\mathbf{W}_l'\mathbf{W}_l)^{-1}\mathbf{1}\|^{-1}\mathbf{1}$, so that each of $s_{j_1}\mathbf{x}_{j_1}, \dots, s_{j_l}\mathbf{x}_{j_l}$ has the same inner product with \mathbf{u}_l . What is not immediately clear (but is argued in Efron, Hastie, Johnstone, and Tibshirani) is why one knows that this prescription agrees with a prescription of a unit vector giving the direction from $\hat{\mathbf{Y}}_{l-1}$ to the projection of \mathbf{Y} onto the sub-space of \mathfrak{R}^N spanned by $\{\mathbf{x}_{j_1}, \dots, \mathbf{x}_{j_l}\}$, namely (for \mathbf{P}_l the projection matrix onto that subspace)

$$\frac{1}{\|\mathbf{P}_l\mathbf{Y} - \hat{\mathbf{Y}}_{l-1}\|} (\mathbf{P}_l\mathbf{Y} - \hat{\mathbf{Y}}_{l-1})$$

Further, the arguments that establish that $|\hat{c}_{l-1,j_1}| = |\hat{c}_{l-1,j_2}| = \dots = |\hat{c}_{l-1,j_l}|$ are not so obvious, nor are those that show that γ_l has the form in 3. above. And finally, HTF actually state their LAR algorithm directly in terms of a path for $\hat{\boldsymbol{\beta}}$, saying at stage l one moves from $\hat{\boldsymbol{\beta}}_{l-1}$ in the direction of a vector with all 0s except at those indices in A where there are the joint least squares coefficients based on the predictor columns $\{\mathbf{x}_{j_1}, \dots, \mathbf{x}_{j_l}\}$. The

correspondence between the two points of view is probably correct, but is again not absolutely obvious.

At any rate, the LAR algorithm traces out a path in \mathfrak{R}^p from $\mathbf{0}$ to $\hat{\boldsymbol{\beta}}^{\text{ols}}$. One might think of the point one has reached along that path (perhaps parameterized by $\|\hat{\mathbf{Y}}\|$) as being a complexity parameter governing how flexible a fit this algorithm has allowed, and be in the business of choosing it (by cross-validation or some other method) in exactly the same way one might, for example, choose a ridge parameter.

What is not at all obvious but true, is that a very slight modification of this LAR algorithm produces the whole set of lasso coefficients (60) as its path. One simply needs to enforce the requirement that if a non-zero coefficient hits 0, its index is *removed* from the active set and a new direction of movement is set based on one less input variable. At any point along the modified LAR path, one can compute $t = \sum_{j=1}^p |\beta_j|$, and think of the modified-LAR path as parameterized by t . (While it's not completely obvious, this turns out to be monotone non-decreasing in "progress along the path," or $\|\hat{\mathbf{Y}}\|$).

A useful graphical representation of the lasso path is one in which all coefficients $\hat{\beta}_{tj}^{\text{lasso}}$ are plotted against t on the same set of axes. Something similar is often done for the LAR coefficients (where the plotting is against some measure of progress along the path defined by the algorithm).

3.2 Two Methods With Derived Input Variables

Another possible approach to finding an appropriate level of complexity in a fitted linear prediction rule is to consider regression on some number $M < p$ of predictors derived from the original inputs x_j . Two such methods are those of Principal Components Regression and Partial Least Squares. Here we continue to assume that **the columns of \mathbf{X} have been standardized and \mathbf{Y} has been centered.**

3.2.1 Principal Components Regression

The idea here is to replace the p columns of predictors in \mathbf{X} with the first few (M) principal components of \mathbf{X} (from the singular value decomposition of \mathbf{X})

$$\mathbf{z}_j = \mathbf{X} \mathbf{v}_j = d_j \mathbf{u}_j$$

Correspondingly, the vector of fitted predictions for the training data is

$$\begin{aligned} \hat{\mathbf{Y}}^{\text{pcr}} &= \sum_{j=1}^M \frac{\langle \mathbf{Y}, \mathbf{z}_j \rangle}{\langle \mathbf{z}_j, \mathbf{z}_j \rangle} \mathbf{z}_j \\ &= \sum_{j=1}^M \langle \mathbf{Y}, \mathbf{u}_j \rangle \mathbf{u}_j \end{aligned} \tag{63}$$

Comparing this to displays (42) and (56) we see that ridge regression shrinks the coefficients of the principal components \mathbf{u}_j according to their importance in making up \mathbf{X} , while principal components regression "zeros out" those least important in making up \mathbf{X} . Further, since the \mathbf{u}_j constitute an orthonormal basis for $C(\mathbf{X})$, for $\text{rank}(\mathbf{X}) = r$,

$$\left\| \widehat{\mathbf{Y}}^{\text{pcr}} \right\|^2 = \sum_{j=1}^M \langle \mathbf{Y}, \mathbf{u}_j \rangle^2 \leq \sum_{j=1}^r \langle \mathbf{Y}, \mathbf{u}_j \rangle^2 = \left\| \widehat{\mathbf{Y}}^{\text{ols}} \right\|^2 \quad (64)$$

Notice too, that $\widehat{\mathbf{Y}}^{\text{pcr}}$ can be written in terms of the original inputs as

$$\begin{aligned} \widehat{\mathbf{Y}}^{\text{pcr}} &= \sum_{j=1}^M \langle \mathbf{Y}, \mathbf{u}_j \rangle \frac{1}{d_j} \mathbf{X} \mathbf{v}_j \\ &= \mathbf{X} \left(\sum_{j=1}^M \langle \mathbf{Y}, \mathbf{u}_j \rangle \frac{1}{d_j} \mathbf{v}_j \right) \\ &= \mathbf{X} \left(\sum_{j=1}^M \frac{1}{d_j^2} \langle \mathbf{Y}, \mathbf{X} \mathbf{v}_j \rangle \mathbf{v}_j \right) \end{aligned}$$

so that

$$\widehat{\boldsymbol{\beta}}^{\text{pcr}} = \sum_{j=1}^M \frac{1}{d_j^2} \langle \mathbf{Y}, \mathbf{X} \mathbf{v}_j \rangle \mathbf{v}_j \quad (65)$$

and $\widehat{\boldsymbol{\beta}}^{\text{ols}}$ is the $M = r = \text{rank}(\mathbf{X})$ version of $\widehat{\boldsymbol{\beta}}^{\text{pcr}}$. As the \mathbf{v}_j are orthonormal, as in relationship (64) above

$$\left\| \widehat{\boldsymbol{\beta}}^{\text{pcr}} \right\| \leq \left\| \widehat{\boldsymbol{\beta}}^{\text{ols}} \right\|$$

and principal components regression shrinks both $\widehat{\mathbf{Y}}^{\text{ols}}$ toward $\mathbf{0}$ in \mathfrak{R}^N and $\widehat{\boldsymbol{\beta}}^{\text{ols}}$ toward $\mathbf{0}$ in \mathfrak{R}^p .

3.2.2 Partial Least Squares Regression

The shrinking methods mentioned thus far have taken no account of \mathbf{Y} in determining directions or amounts of shrinkage. Partial least squares specifically employs \mathbf{Y} . In what follows, we continue to suppose that the columns of \mathbf{X} have been standardized and that \mathbf{Y} has been centered.

The logic of partial least squares is this. Suppose that

$$\begin{aligned} \mathbf{z}_1 &= \sum_{j=1}^p \langle \mathbf{Y}, \mathbf{x}_j \rangle \mathbf{x}_j \\ &= \mathbf{X} \mathbf{X}' \mathbf{Y} \end{aligned}$$

It is possible to argue that for $\mathbf{w}_1 = \mathbf{X}'\mathbf{Y}/\|\mathbf{X}'\mathbf{Y}\|$, $\mathbf{X}\mathbf{w}_1 = \mathbf{z}_1/\|\mathbf{X}'\mathbf{Y}\|$ is a linear combination of the columns of \mathbf{X} maximizing

$$|\langle \mathbf{Y}, \mathbf{X}\mathbf{w} \rangle|$$

(which is essentially the absolute sample covariance between the variables y and $\mathbf{x}'\mathbf{w}$) subject to the constraint that $\|\mathbf{w}\| = 1$.¹⁷ This follows because

$$\langle \mathbf{Y}, \mathbf{X}\mathbf{w} \rangle^2 = \mathbf{w}'\mathbf{X}'\mathbf{Y}\mathbf{Y}'\mathbf{X}\mathbf{w}$$

and a maximizer of this quadratic form subject to the constraint is the eigenvector of $\mathbf{X}'\mathbf{Y}\mathbf{Y}'\mathbf{X}$ corresponding to its single non-zero eigenvalue. It's then easy to verify that \mathbf{w}_1 is such an eigenvector corresponding to the non-zero eigenvalue $\mathbf{Y}'\mathbf{X}\mathbf{X}'\mathbf{Y}$.

Then define \mathbf{X}^1 by orthogonalizing the columns of \mathbf{X} with respect to \mathbf{z}_1 . That is, define the j th column of \mathbf{X}^1 by

$$\mathbf{x}_j^1 = \mathbf{x}_j - \frac{\langle \mathbf{x}_j, \mathbf{z}_1 \rangle}{\langle \mathbf{z}_1, \mathbf{z}_1 \rangle} \mathbf{z}_1$$

and take

$$\begin{aligned} \mathbf{z}_2 &= \sum_{j=1}^p \langle \mathbf{Y}, \mathbf{x}_j^1 \rangle \mathbf{x}_j^1 \\ &= \mathbf{X}^1 \mathbf{X}^{1'} \mathbf{Y} \end{aligned}$$

For $\mathbf{w}_2 = \mathbf{X}^{1'}\mathbf{Y}/\|\mathbf{X}^{1'}\mathbf{Y}\|$, $\mathbf{X}^1\mathbf{w}_2 = \mathbf{z}_2/\|\mathbf{X}^{1'}\mathbf{Y}\|$ is the linear combination of the columns of \mathbf{X}^1 maximizing

$$|\langle \mathbf{Y}, \mathbf{X}^1\mathbf{w} \rangle|$$

subject to the constraint that $\|\mathbf{w}\| = 1$.

Then for $l > 1$, define \mathbf{X}^l by orthogonalizing the columns of \mathbf{X}^{l-1} with respect to \mathbf{z}_l . That is, define the j th column of \mathbf{X}^l by

$$\mathbf{x}_j^l = \mathbf{x}_j^{l-1} - \frac{\langle \mathbf{x}_j^{l-1}, \mathbf{z}_l \rangle}{\langle \mathbf{z}_l, \mathbf{z}_l \rangle} \mathbf{z}_l$$

and let

$$\begin{aligned} \mathbf{z}_{l+1} &= \sum_{j=1}^p \langle \mathbf{Y}, \mathbf{x}_j^l \rangle \mathbf{x}_j^l \\ &= \mathbf{X}^l \mathbf{X}^{l'} \mathbf{Y} \end{aligned}$$

¹⁷Note that upon replacing $|\langle \mathbf{Y}, \mathbf{X}\mathbf{w} \rangle|$ with $|\langle \mathbf{X}\mathbf{w}, \mathbf{X}\mathbf{w} \rangle|$ one has the kind of optimization problem solved by the first principal component of \mathbf{X} .

Partial least squares regression uses the first M of these variables \mathbf{z}_j as input variables.

The PLS predictors \mathbf{z}_j are orthogonal by construction. Using the first M of these as regressors, one has the vector of fitted output values

$$\widehat{\mathbf{Y}}^{\text{pls}} = \sum_{j=1}^M \frac{\langle \mathbf{Y}, \mathbf{z}_j \rangle}{\langle \mathbf{z}_j, \mathbf{z}_j \rangle} \mathbf{z}_j$$

Since the PLS predictors are (albeit recursively-computed data-dependent) linear combinations of columns of \mathbf{X} , it is possible to find a p -vector $\widehat{\boldsymbol{\beta}}_M^{\text{pls}}$ (namely $(\mathbf{X}'\mathbf{X})^{-1} \mathbf{X}'\widehat{\mathbf{Y}}^{\text{pls}}$) such that

$$\widehat{\mathbf{Y}}^{\text{pls}} = \mathbf{X}\widehat{\boldsymbol{\beta}}_M^{\text{pls}}$$

and thus produce the corresponding linear prediction rule

$$\widehat{f}(\mathbf{x}) = \mathbf{x}'\widehat{\boldsymbol{\beta}}_M^{\text{pls}} \quad (66)$$

It is tempting to think that in form (66), the number of components, M , should function as a complexity parameter. But then again there is the following. When the \mathbf{x}_j are orthogonal, it's fairly easy to see that \mathbf{z}_1 is a multiple of $\widehat{\mathbf{Y}}^{\text{ols}}$. That is, in this circumstance,

$$\mathbf{X}'\mathbf{X} = N\mathbf{I}$$

so that

$$\widehat{\mathbf{Y}}^{\text{ols}} = \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1} \mathbf{X}'\mathbf{Y} = \frac{1}{N} \mathbf{X}\mathbf{X}'\mathbf{Y} = \frac{1}{N} \mathbf{z}_1$$

i.e.

$$\mathbf{z}_1 = N\widehat{\mathbf{Y}}^{\text{ols}}$$

so that

$$\widehat{\mathbf{Y}}_1^{\text{pls}} = \widehat{\mathbf{Y}}^{\text{ols}}$$

and thus $\widehat{\boldsymbol{\beta}}_1^{\text{pls}} = \widehat{\boldsymbol{\beta}}_2^{\text{pls}} = \dots = \widehat{\boldsymbol{\beta}}_p^{\text{pls}} = \widehat{\boldsymbol{\beta}}^{\text{ols}}$. All steps of partial least squares after the first are simply providing a basis for the orthogonal complement of the 1-dimensional subspace of $C(\mathbf{X})$ generated by $\widehat{\mathbf{Y}}^{\text{ols}}$ (without improving fitting at all). That is, here changing M doesn't change flexibility of the fit at all. (Presumably, when the \mathbf{x}_j are nearly orthogonal, something similar happens.)

This observation about PLS in cases where predictors are orthogonal has another related implication. That is that there will be no naive form for effective degrees of freedom for PLS. Since with \mathbf{z}_j the j th principal component of \mathbf{X} and, say,

$$\mathbf{Z}^M = (\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_M)$$

we have

$$\widehat{\mathbf{Y}}^{\text{PCR}} = \mathbf{Z}^M \left((\mathbf{Z}^M)' \mathbf{Z}^M \right)^{-1} (\mathbf{Z}^M)' \mathbf{Y}$$

principal components regression on M components has effective degrees of freedom M . But the fact that the " \mathbf{Z}^M " matrix corresponding to PLS depends upon \mathbf{Y} makes PLS *nonlinear in \mathbf{Y}* . And the "orthogonal \mathbf{X} " argument shows that a PLS predictor with $M = 1$ can have effective degrees of freedom as large as $\text{rank}(\mathbf{X})$.

PLS, PCR, and OLS Partial least squares is a kind of compromise between principal components regression and ordinary least squares. To see this, note that maximizing

$$|\langle \mathbf{Y}, \mathbf{X} \mathbf{w} \rangle|$$

subject to the constraint that $\|\mathbf{w}\| = 1$ is equivalent to maximizing the absolute sample covariance between \mathbf{Y} and $\mathbf{X} \mathbf{w}$ i.e.

$$\left(\begin{array}{c} \text{sample standard} \\ \text{deviation of } y \end{array} \right) \cdot \left(\begin{array}{c} \text{sample standard} \\ \text{deviation of } \mathbf{x}'\mathbf{w} \end{array} \right) \cdot \left| \left(\begin{array}{c} \text{sample correlation} \\ \text{between } y \text{ and } \mathbf{x}'\mathbf{w} \end{array} \right) \right|$$

or equivalently

$$\left(\begin{array}{c} \text{sample variance} \\ \text{of } \mathbf{x}'\mathbf{w} \end{array} \right) \cdot \left(\begin{array}{c} \text{sample correlation} \\ \text{between } y \text{ and } \mathbf{x}'\mathbf{w} \end{array} \right)^2 \quad (67)$$

subject to the constraint. Now if only the first term (the sample variance of $\mathbf{x}'\mathbf{w}$) were involved in product (67), a first principal component direction would be an optimizing \mathbf{w}_1 , and $\mathbf{z}_1 = \|\mathbf{X}'\mathbf{Y}\| \mathbf{X} \mathbf{w}_1$ a multiple of the first principal component of \mathbf{X} . On the other hand, if only the second term were involved, $\widehat{\boldsymbol{\beta}}^{\text{ols}} / \|\widehat{\boldsymbol{\beta}}^{\text{ols}}\|$ would be an optimizing \mathbf{w}_1 , and $\mathbf{z}_1 = \widehat{\mathbf{Y}}^{\text{ols}} \|\mathbf{X}'\mathbf{Y}\| / \|\widehat{\boldsymbol{\beta}}^{\text{ols}}\|$ a multiple of the vector of ordinary least squares fitted values. The use of the product of two terms can be expected to produce a compromise between these two.

Note further that this logic applied at later steps in the PLS algorithm then produces for \mathbf{z}_l a compromise between a first principal component of \mathbf{X}^{l-1} and a suitably constrained multiple of the vector of least squares fitted values based on the matrix of inputs \mathbf{X}^{l-1} . The matrices \mathbf{X}^l have columns that are the projections of the corresponding columns of \mathbf{X} onto the orthogonal complement in $\mathcal{C}(\mathbf{X})$ of the span of $\{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_l\}$ (i.e. are corresponding columns of \mathbf{X} minus their projections onto the span of $\{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_l\}$) and $\mathcal{C}(\mathbf{X}) \supset \mathcal{C}(\mathbf{X}^1) \supset \mathcal{C}(\mathbf{X}^2) \dots$.

4 Linear SEL Prediction Using Basis Functions

A way of moving beyond prediction rules that are functions of a linear form in \mathbf{x} , i.e. depend upon \mathbf{x} through $\mathbf{x}'\widehat{\boldsymbol{\beta}}$, is to consider some set of (basis)¹⁸ functions

¹⁸The word "basis" is employed here to point to the notion of a "basis" in a linear space of functions, whereby any function of interest can be represented (or practically speaking, at

$\{h_m\}$ and predictors of the form or depending upon the form

$$\hat{f}(\mathbf{x}) = \sum_{m=1}^p \hat{\beta}_m h_m(\mathbf{x}) = \mathbf{h}(\mathbf{x})' \hat{\boldsymbol{\beta}} \quad (68)$$

for $\mathbf{h}(\mathbf{x})' = (h_1(\mathbf{x}), \dots, h_p(\mathbf{x}))$. (The general notation used in Section 1.4.5 was $T(\mathbf{x})$ rather than $\mathbf{h}(\mathbf{x})$ being used here. The slight specialization here is to the case where the components of the vector-valued $\mathbf{h}(\mathbf{x})$ are "basis" functions.)

We next consider some flexible methods employing this idea. Notice that fitting of form (68) can be done using any of the methods just discussed based on the $N \times p$ matrix of inputs

$$\mathbf{X} = (h_j(\mathbf{x}_i)) = \begin{pmatrix} \mathbf{h}(\mathbf{x}_1)' \\ \mathbf{h}(\mathbf{x}_2)' \\ \vdots \\ \mathbf{h}(\mathbf{x}_N)' \end{pmatrix}$$

(i indexing rows and j indexing columns).

4.1 $p = 1$ Wavelet Bases

Consider first the case of a one-dimensional input variable x , and in fact here suppose that x takes values in $[0, 1]$. One might consider a set of basis function for use in the form (68) that is big enough and rich enough to approximate essentially any function on $[0, 1]$. In particular, various orthonormal bases for the square integrable functions on this interval (the space of functions $L_2[0, 1]$) come to mind. One might, for example, consider using some number of functions from the Fourier basis for $L_2[0, 1]$

$$\left\{ \sqrt{2} \sin(j2\pi x) \right\}_{j=1}^{\infty} \cup \left\{ \sqrt{2} \cos(j2\pi x) \right\}_{j=1}^{\infty} \cup \{1\}$$

For example, using $M \approx N/2$ sin-cos pairs and the constant, one could consider the fitting the forms

$$f(x) = \beta_0 + \sum_{m=1}^M \beta_{1m} \sin(m2\pi x) + \sum_{m=1}^M \beta_{2m} \cos(m2\pi x) \quad (69)$$

If one has training x_i on an appropriate regular grid, the use of form (69) leads to orthogonality in the $N \times (2M + 1)$ matrix of values of the basis functions \mathbf{X} and simple/fast calculations.

least approximated) as a linear combination of the the "basis" elements. Periodic functions of a single variable can be approximated by linear combinations of sine (basis) functions of various frequencies. General differentiable functions can be approximated by polynomials (linear combinations of monomial basis functions). Etc.

Unless, however, one believes that $E[y|x = u]$ is periodic in u , form (69) has its serious limitations. In particular, unless M is very very large, a trigonometric series like (69) will typically provide a poor approximation for a function that varies at different scales on different parts of $[0, 1]$, and in any case, the coefficients necessary to provide such localized variation at different scales have no obvious simple interpretations/connections to the irregular pattern of variation being described. So-called "wavelet bases" are much more useful in providing parsimonious and interpretable approximations to such functions. The simplest wavelet basis for $L_2[0, 1]$ is the Haar basis that we proceed to describe.

Define the so-called Haar "**father**" wavelet

$$\varphi(x) = I[0 < x \leq 1]$$

and the so-called Haar "**mother**" wavelet

$$\begin{aligned}\psi(x) &= \varphi(2x) - \varphi(2x - 1) \\ &= I\left[0 < x \leq \frac{1}{2}\right] - I\left[\frac{1}{2} < x \leq 1\right]\end{aligned}$$

Linear combinations of these functions provide all elements of $L_2[0, 1]$ that are constant on $(0, \frac{1}{2}]$ and on $(\frac{1}{2}, 1]$. Write

$$\Psi_0 = \{\varphi, \psi\}$$

Next, define

$$\begin{aligned}\psi_{1,0}(x) &= \sqrt{2} \left(I\left[0 < x \leq \frac{1}{4}\right] - I\left[\frac{1}{4} < x \leq \frac{1}{2}\right] \right) \quad \text{and} \\ \psi_{1,1}(x) &= \sqrt{2} \left(I\left[\frac{1}{2} < x \leq \frac{3}{4}\right] - I\left[\frac{3}{4} < x \leq 1\right] \right)\end{aligned}$$

and let

$$\Psi_1 = \{\psi_{1,0}, \psi_{1,1}\}$$

Using the set of functions $\Psi_0 \cup \Psi_1$ one can build (as linear combinations) all elements of $L_2[0, 1]$ that are constant on $(0, \frac{1}{4}]$ and on $(\frac{1}{4}, \frac{1}{2}]$ and on $(\frac{1}{2}, \frac{3}{4}]$ and on $(\frac{3}{4}, 1]$.

The story then goes on as one should expect. One defines

$$\begin{aligned}\psi_{2,0}(x) &= 2 \left(I\left[0 < x \leq \frac{1}{8}\right] - I\left[\frac{1}{8} < x \leq \frac{1}{4}\right] \right) \quad \text{and} \\ \psi_{2,1}(x) &= 2 \left(I\left[\frac{1}{4} < x \leq \frac{3}{8}\right] - I\left[\frac{3}{8} < x \leq \frac{1}{2}\right] \right) \quad \text{and} \\ \psi_{2,2}(x) &= 2 \left(I\left[\frac{1}{2} < x \leq \frac{5}{8}\right] - I\left[\frac{5}{8} < x \leq \frac{3}{4}\right] \right) \quad \text{and} \\ \psi_{2,3}(x) &= 2 \left(I\left[\frac{3}{4} < x \leq \frac{7}{8}\right] - I\left[\frac{7}{8} < x \leq 1\right] \right)\end{aligned}$$

and lets

$$\Psi_2 = \{\psi_{2,0}, \psi_{2,1}, \psi_{2,2}, \psi_{2,3}\}$$

Figure 21 shows the sets of basis functions Ψ_0 , Ψ_1 , and Ψ_2 .

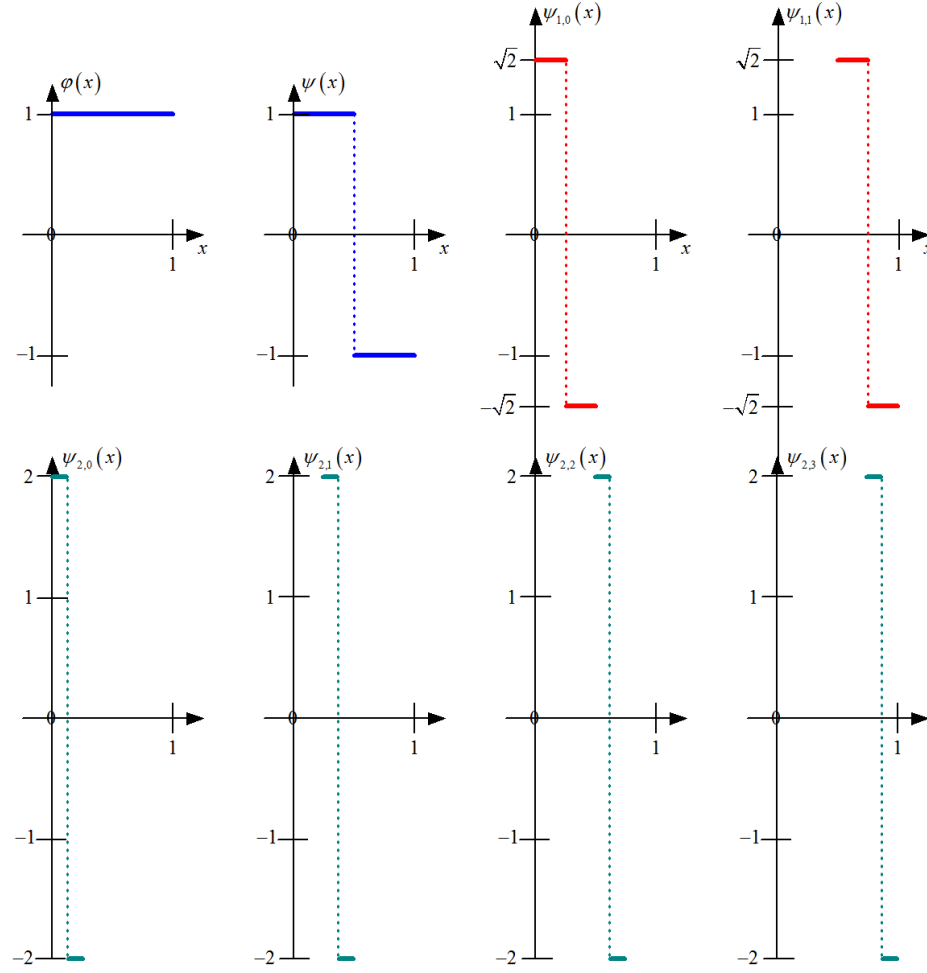


Figure 21: Sets of Haar basis functions Ψ_0 (blue), Ψ_1 (red), and Ψ_2 (green).

In general,

$$\psi_{m,j}(x) = \sqrt{2^m} \psi \left(2^m \left(x - \frac{j}{2^m} \right) \right) \quad \text{for } j = 0, 1, 2, \dots, 2^m - 1$$

and

$$\Psi_m = \{\psi_{m,0}, \psi_{m,1}, \dots, \psi_{m,2^m-1}\}$$

The Haar basis of $L_2[0, 1]$ is then

$$\cup_{m=0}^{\infty} \Psi_m$$

Then, one might entertain use of the Haar basis functions through order M in constructing a form

$$f(x) = \beta_0 + \sum_{m=0}^M \sum_{j=0}^{2^m-1} \beta_{mj} \psi_{m,j}(x) \quad (70)$$

(with the understanding that $\psi_{0,0} = \psi$), a form that in general allows building of functions that are constant on consecutive intervals of length $1/2^{M+1}$. This form can be fit by any of the various regression methods (especially involving thresholding/selection, as a typically very large number, 2^{M+1} , of basis functions is employed in form (70)). (See HTF Section 5.9.2 for some discussion of using the lasso with wavelets.) Large absolute values of coefficients β_{mj} encode scales at which important variation in the value of the index m , and location in $[0, 1]$ where that variation occurs in the value $j/2^m$. Where (perhaps after model selection/ thresholding) only a relatively few fitted coefficients are important, the corresponding scales and locations provide an informative and compact summary of the fit. A nice visual summary of the results of the fit can be made by plotting for each m (plots arranged vertically, from M through 0, aligned and to the same scale) spikes of length $|\beta_{mj}|$ pointed in the direction of $\text{sign}(\beta_{mj})$ along an " x " axis at positions (say) $(j/2^m) + 1/2^{m+1}$.

In special situations where $N = 2^K$ and

$$x_i = i \left(\frac{1}{2^K} \right) \text{ for } i = 1, 2, \dots, 2^K$$

and one uses the Haar basis functions through order $K - 1$, the fitting of form (70) is computationally clean, since the vectors

$$\begin{pmatrix} \psi_{m,j}(x_1) \\ \vdots \\ \psi_{m,j}(x_N) \end{pmatrix}$$

(together with the column vector of 1s) are orthogonal. (So, upon proper normalization, i.e. division by $\sqrt{N} = 2^{K/2}$, they form an orthonormal basis for \mathfrak{R}^N .)

The Haar wavelet basis functions are easy to describe and understand. But they are discontinuous, and from some points of view that is unappealing. Other sets of wavelet basis functions have been developed that are smooth. The construction begins with a smooth "mother wavelet" in place of the step function used above. HTF make some discussion of the smooth "symmlet" wavelet basis at the end of their Chapter 5.

4.2 $p = 1$ Piecewise Polynomials and Regression Splines

Continue consideration of the case of a one-dimensional input variable x , and now K "knots"

$$\xi_1 < \xi_2 < \dots < \xi_K$$

and forms for $f(x)$ that are

1. polynomials of order M (or less) on all intervals (ξ_{j-1}, ξ_j) , and (potentially, at least)
2. have derivatives of some specified order at the knots, and (potentially, at least)
3. are linear outside (ξ_1, ξ_K) .

If we let $I_1(x) = I[x < \xi_1]$, for $j = 2, \dots, K$ let $I_j(x) = I[\xi_{j-1} \leq x < \xi_j]$, and define $I_{K+1}(x) = I[\xi_K \leq x]$, one can have 1. in the list above using basis functions

$$\begin{aligned}
& I_1(x), I_2(x), \dots, I_{K+1}(x) \\
& xI_1(x), xI_2(x), \dots, xI_{K+1}(x) \\
& x^2I_1(x), x^2I_2(x), \dots, x^2I_{K+1}(x) \\
& \vdots \\
& x^M I_1(x), x^M I_2(x), \dots, x^M I_{K+1}(x)
\end{aligned}$$

Further, one can enforce continuity and differentiability (at the knots) conditions on a form $f(x) = \sum_{m=1}^{(M+1)(K+1)} \beta_m h_m(x)$ by enforcing some linear relations between appropriate ones of the β_m . While this is conceptually simple, it is messy. It is much cleaner to simply begin with a set of basis functions that are tailored to have the desired continuity/differentiability properties.

A set of $M + 1 + K$ basis functions for piecewise polynomials of degree M with derivatives of order $M - 1$ at all knots is easily seen to be

$$1, x, x^2, \dots, x^M, (x - \xi_1)_+^M, (x - \xi_2)_+^M, \dots, (x - \xi_K)_+^M$$

(since the value and first $M - 1$ derivatives of $(x - \xi_j)_+^M$ at ξ_j are all 0). The choice of $M = 3$ is fairly standard.

Since extrapolation with polynomials typically gets worse with order, it is common to impose a restriction that outside (ξ_1, ξ_K) a form $f(x)$ be linear. For the case of $M = 3$ this can be accomplished by beginning with basis functions $1, x, (x - \xi_1)_+^3, (x - \xi_2)_+^3, \dots, (x - \xi_K)_+^3$ and imposing restrictions necessary to force 2nd and 3rd derivatives to the right of ξ_K to be 0. Notice that (considering $x > \xi_K$)

$$\frac{d^2}{dx^2} \left(\alpha_0 + \alpha_1 x + \sum_{j=1}^K \beta_j (x - \xi_j)_+^3 \right) = 6 \sum_{j=1}^K \beta_j (x - \xi_j) \quad (71)$$

and

$$\frac{d^3}{dx^3} \left(\alpha_0 + \alpha_1 x + \sum_{j=1}^K \beta_j (x - \xi_j)_+^3 \right) = 6 \sum_{j=1}^K \beta_j \quad (72)$$

So, linearity for large x requires (from equation (72)) that $\sum_{j=1}^K \beta_j = 0$. Further, substituting this into relationship (71) means that linearity also requires that $\sum_{j=1}^K \beta_j \xi_j = 0$. Using the first of these to conclude that $\beta_K = -\sum_{j=1}^{K-1} \beta_j$ and substituting into the second yields

$$\beta_{K-1} = -\sum_{j=1}^{K-2} \beta_j \left(\frac{\xi_K - \xi_j}{\xi_K - \xi_{K-1}} \right)$$

and then

$$\beta_K = \sum_{j=1}^{K-2} \beta_j \left(\frac{\xi_K - \xi_j}{\xi_K - \xi_{K-1}} \right) - \sum_{j=1}^{K-2} \beta_j$$

These then suggest the set of basis functions consisting of $1, x$ and for $j = 1, 2, \dots, K-2$

$$\begin{aligned} (x - \xi_j)_+^3 - \left(\frac{\xi_K - \xi_j}{\xi_K - \xi_{K-1}} \right) (x - \xi_{K-1})_+^3 + \left(\frac{\xi_K - \xi_j}{\xi_K - \xi_{K-1}} \right) (x - \xi_K)_+^3 - (x - \xi_K)_+^3 \\ (73) \\ = (x - \xi_j)_+^3 - \left(\frac{\xi_K - \xi_j}{\xi_K - \xi_{K-1}} \right) (x - \xi_{K-1})_+^3 + \left(\frac{\xi_{K-1} - \xi_j}{\xi_K - \xi_{K-1}} \right) (x - \xi_K)_+^3 \end{aligned}$$

(These are essentially the basis functions that HTF call their N_j .) Their use produces so-called "natural" (linear outside (ξ_1, ξ_K)) cubic regression splines.

There are other (harder to motivate, but in the end more pleasing and computationally more attractive) sets of basis functions for natural polynomial splines. See the B-spline material at the end of HTF Chapter 5.

4.3 Basis Functions and p -Dimensional Inputs

4.3.1 Multi-Dimensional Regression Splines (Tensor Product Bases)

If $p = 2$ and the vector of inputs, \mathbf{x} , takes values in \mathfrak{R}^2 , one might proceed as follows. If $\{h_{11}, h_{12}, \dots, h_{1M_1}\}$ is a set of spline basis functions based on x_1 and $\{h_{21}, h_{22}, \dots, h_{2M_2}\}$ is a set of spline basis functions based on x_2 one might consider the set of $M_1 \cdot M_2$ basis functions based on \mathbf{x} defined by

$$g_{jk}(\mathbf{x}) = h_{1j}(x_1) h_{2k}(x_2)$$

and corresponding forms for regression splines

$$f(\mathbf{x}) = \sum_{j,k} \beta_{jk} g_{jk}(\mathbf{x}) \quad (74)$$

The biggest problem with this potential method is the explosion in the size of a tensor product basis as p increases. For example, using K knots for cubic regression splines in each of p dimensions produces $(4 + K)^p$ basis functions for the p -dimensional problem. Some kind of forward selection algorithm or

shrinking of coefficients will be needed to produce any kind of workable fits with such large numbers of basis functions. For example, the multivariate smoothing routines provided in the `mgcv` R package of Wood allow for quadratic penalized (ridge regression type) fitting of forms like (74). The following discussion of "MARS" concerns one kind of forward selection algorithm using (data-dependent) linear regression spline basis functions and products of them for building predictors

4.3.2 MARS (Multivariate Adaptive Regression Splines)

This is a high-dimensional regression methodology based on use of data-dependent "hockey-stick" or "hinge" functions (the kind of functions leading to piece-wise linear regression splines when $p = 1$) and their products as (data-dependent) "basis functions." That is, with input space \mathcal{R}^p consider defining *data-dependent* features¹⁹ built on the Np pairs of functions

$$h_{ij1}(\mathbf{x}) = (x_j - x_{ij})_+ \quad \text{and} \quad h_{ij2}(\mathbf{x}) = (x_{ij} - x_j)_+ \quad (75)$$

(x_{ij} is the j th coordinate of the i th input training vector and both $h_{ij1}(\mathbf{x})$ and $h_{ij2}(\mathbf{x})$ depend on \mathbf{x} only through the j th coordinate of \mathbf{x}) portrayed in Figure 22.

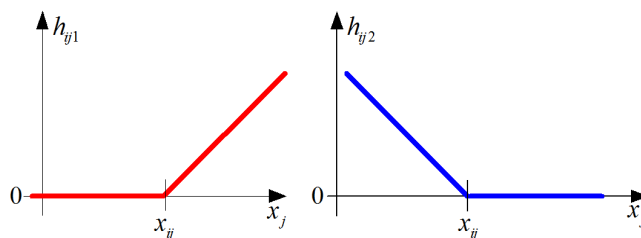


Figure 22: Pair of hinge functions.

MARS builds predictors sequentially, making use of these "reflected pairs" of hinge functions and their products. One version (described in HTF) proceeds roughly as follows.

1. Identify a pair (75) so that

$$\beta_0 + \beta_{11}h_{ij1}(\mathbf{x}) + \beta_{12}h_{ij2}(\mathbf{x})$$

has the best SSE possible. Call the selected functions

$$g_{11} = h_{ij1} \quad \text{and} \quad g_{12} = h_{ij2}$$

and set

$$\hat{f}_1(\mathbf{x}) = \hat{\beta}_0 + \hat{\beta}_{11}g_{11}(\mathbf{x}) + \hat{\beta}_{12}g_{12}(\mathbf{x})$$

¹⁹Notice that in the framework of Section 1.4.5 these functions of the input \mathbf{x} are of the form $T(\mathbf{T}, \mathbf{x})$, NOT simply of the form $T(\mathbf{x})$.

2. At stage l of the predictor-building process, with predictor

$$\hat{f}_{l-1}(\mathbf{x}) = \hat{\beta}_0 + \sum_{m=1}^{l-1} \left(\hat{\beta}_{m1} g_{m1}(\mathbf{x}) + \hat{\beta}_{m2} g_{m2}(\mathbf{x}) \right)$$

in hand, consider for addition to the model pairs of functions that are either of the basic form (75) or of the form

$$h_{ij1}(\mathbf{x}) g_{m1}(\mathbf{x}) \text{ and } h_{ij2}(\mathbf{x}) g_{m1}(\mathbf{x})$$

or of the form

$$h_{ij1}(\mathbf{x}) g_{m2}(\mathbf{x}) \text{ and } h_{ij2}(\mathbf{x}) g_{m2}(\mathbf{x})$$

for some $m < l$, subject to the constraint that no x_j appears in any candidate product more than once (maintaining the piece-wise linearity of sections of the predictor). Additionally, one may decide to put an upper limit on the order of the products considered for inclusion in the predictor. The best candidate pair in terms of reducing SSE gets called, say, g_{l1} and g_{l2} and one sets

$$\hat{f}_l(\mathbf{x}) = \hat{\beta}_0 + \sum_{m=1}^l \left(\hat{\beta}_{m1} g_{m1}(\mathbf{x}) + \hat{\beta}_{m2} g_{m2}(\mathbf{x}) \right)$$

One might pick the complexity parameter l by cross-validation, but the standard implementation of MARS apparently uses instead a kind of generalized cross validation error

$$GCV(l) = \frac{\sum_{i=1}^N (y_i - \hat{f}_l(\mathbf{x}_i))^2}{\left(1 - \frac{M(l)}{N}\right)^2}$$

where $M(l)$ is some kind of degrees of freedom figure. One must take account of both the fitting of the coefficients β in this and the fact that knots (values x_{ij}) have been chosen. The HTF recommendation is to use

$$M(l) = 2l + (2 \text{ or } 3) \cdot (\text{the number of different knots chosen})$$

(where presumably the knot count refers to different x_{ij} appearing in at least one $g_{m1}(\mathbf{x})$ or $g_{m2}(\mathbf{x})$).

Other versions of "MARS" algorithms potentially remove the constraint that no x_j appear in any candidate product more than once (eliminating the piece-wise linearity of sections of the predictor), consider not pairs but single hinge functions at each stage of feature addition, and/or follow a forward-selection search for features with a backwards-elimination phase (these guided by significant "change in SSE" or "F/t test" criteria). All of these variants amount

to the "special sauce" of a particular MARS implementation set by its designer/programmer. Particular implementations have user-selectable parameters like the maximum number of terms in a forward selection phase, the maximum order of (pure and mixed) terms considered, the "significance level" used for guiding forward and backward phases of selection of "features," etc. In practical application, one should select these parameters via cross-validation, more or less thinking of whatever choices the developer has made in his or her implementation as simply defining some fitting/predictor-building "black box." A routine like the `train()` function in `caret` is invaluable in making these choices.

Figure 23 portrays a simple predictor (of home sales price) of the kind that a MARS algorithm can produce.

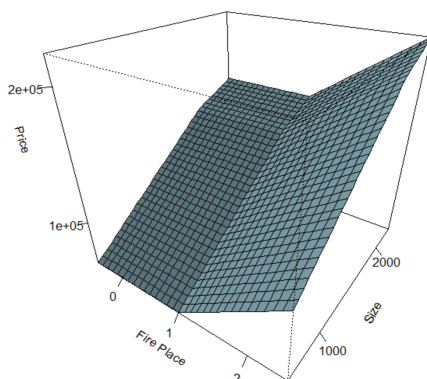


Figure 23: An example of the kind of prediction surface that can be generated by a MARS algorithm. "Price" varies with two predictors.

5 Smoothing Splines and SEL Prediction

5.1 $p = 1$ Smoothing Splines

A way of avoiding the direct selection of knots for a regression spline is to instead, for a smoothing parameter $\lambda > 0$, consider the problem of finding (for $a \leq \min \{x_i\}$ and $\max \{x_i\} \leq b$)

$$\hat{f}_\lambda = \underset{\text{functions } h \text{ with 2 derivatives}}{\arg \min} \left(\sum_{i=1}^N (y_i - h(x_i))^2 + \lambda \int_a^b (h''(x))^2 dx \right)$$

Amazingly enough, this optimization problem has a solution that can be fairly simply described. \hat{f}_λ is a natural cubic spline with knots at the distinct values x_i in the training set. That is, for a set of (now data-dependent, as the knots come from the training data) basis functions for such splines

$$h_1, h_2, \dots, h_N$$

(here we're tacitly assuming that the N values of the input variable in the training set are all different)

$$\hat{f}_\lambda(x) = \sum_{j=1}^N \hat{\beta}_{\lambda j} h_j(x) \quad (76)$$

where the $\hat{\beta}_{\lambda j}$ are yet to be identified.

So consider the function

$$g(x) = \sum_{j=1}^N \theta_j h_j(x) \quad (77)$$

This has second derivative

$$g''(x) = \sum_{j=1}^N \theta_j h_j''(x)$$

and so

$$(g''(x))^2 = \sum_{j=1}^N \sum_{l=1}^N \theta_j \theta_l h_j''(x) h_l''(x)$$

Then, for $\boldsymbol{\theta}' = (\theta_1, \theta_2, \dots, \theta_N)$ and²⁰

$$\boldsymbol{\Omega}_{N \times N} = \left(\int_a^b h_j''(t) h_l''(t) dt \right)$$

it is the case that

$$\int_a^b (g''(x))^2 dx = \boldsymbol{\theta}' \boldsymbol{\Omega} \boldsymbol{\theta}$$

In fact, with the notation

$$\mathbf{H}_{N \times N} = (h_j(x_i))$$

(i indexing rows and j indexing columns) the criterion to be optimized in order to find \hat{f}_λ can be written for functions of the form (77) as

$$(\mathbf{Y} - \mathbf{H}\boldsymbol{\theta})' (\mathbf{Y} - \mathbf{H}\boldsymbol{\theta}) + \lambda \boldsymbol{\theta}' \boldsymbol{\Omega} \boldsymbol{\theta}$$

and some vector calculus shows that the optimizing $\boldsymbol{\theta}$ is

$$\hat{\boldsymbol{\beta}}_\lambda = (\mathbf{H}'\mathbf{H} + \lambda \boldsymbol{\Omega})^{-1} \mathbf{H}'\mathbf{Y} \quad (78)$$

which can be thought of as some kind of vector of generalized ridge regression coefficients. This form (78) together with representation (76) of course provides a smoothed prediction of y for *any* input x .

²⁰For the set of cubic spline basis functions (73) it is unpleasant but straightforward to find relatively simple formulas for the entries of $\boldsymbol{\Omega}$. See the exercises for this section for details.

Corresponding to coefficient vector (78) is a vector of smoothed output values

$$\hat{\mathbf{Y}}_\lambda = \mathbf{H} (\mathbf{H}'\mathbf{H} + \lambda\boldsymbol{\Omega})^{-1} \mathbf{H}' \mathbf{Y}$$

and the matrix

$$\mathbf{S}_\lambda \equiv \mathbf{H} (\mathbf{H}'\mathbf{H} + \lambda\boldsymbol{\Omega})^{-1} \mathbf{H}'$$

is called a smoother matrix.

Contrast this to a situation where some fairly small number, p , of *fixed* basis functions are employed in a regression context. That is, for basis functions b_1, b_2, \dots, b_p suppose

$$\underset{N \times p}{\mathbf{B}} = (b_j(x_i))$$

Then OLS produces the vector of fitted values

$$\hat{\mathbf{Y}} = \mathbf{B} (\mathbf{B}'\mathbf{B})^{-1} \mathbf{B}' \mathbf{Y}$$

and the projection matrix onto the column space of \mathbf{B} , $C(\mathbf{B})$, is $\mathbf{P}_B = \mathbf{B} (\mathbf{B}'\mathbf{B})^{-1} \mathbf{B}'$.

\mathbf{S}_λ and \mathbf{P}_B are both $N \times N$ symmetric non-negative definite matrices. While

$$\mathbf{P}_B \mathbf{P}_B = \mathbf{P}_B$$

i.e. \mathbf{P}_B is idempotent,

$$\mathbf{S}_\lambda \mathbf{S}_\lambda \preceq \mathbf{S}_\lambda$$

meaning that $\mathbf{S}_\lambda - \mathbf{S}_\lambda \mathbf{S}_\lambda$ is non-negative definite. \mathbf{P}_B is of rank $p = \text{tr}(\mathbf{P}_B)$, while \mathbf{S}_λ is of rank N .

In a manner similar to what is done in ridge regression we might define an "effective degrees of freedom" for \mathbf{S}_λ (or for smoothing) as

$$\text{df}(\lambda) = \text{tr}(\mathbf{S}_\lambda) \tag{79}$$

We proceed to develop motivation and a formula for this quantity and for $\hat{\mathbf{Y}}_\lambda$. Notice that for

$$\mathbf{K} = (\mathbf{H}')^{-1} \boldsymbol{\Omega} \mathbf{H}^{-1}$$

one has

$$\begin{aligned} \mathbf{S}_\lambda &= \mathbf{H} (\mathbf{H}'\mathbf{H} + \lambda\boldsymbol{\Omega})^{-1} \mathbf{H}' \\ &= \mathbf{H} (\mathbf{H}' (\mathbf{I} + \lambda\mathbf{H}'^{-1}\boldsymbol{\Omega}\mathbf{H}) \mathbf{H})^{-1} \mathbf{H}' \\ &= \mathbf{H}\mathbf{H}^{-1} (\mathbf{I} + \lambda\mathbf{H}'^{-1}\boldsymbol{\Omega}\mathbf{H})^{-1} \mathbf{H}'^{-1} \mathbf{H}' \\ &= (\mathbf{I} + \lambda\mathbf{K})^{-1} \end{aligned} \tag{80}$$

This is the so-called **Reinsch form** for \mathbf{S}_λ , from whence $\mathbf{S}_\lambda^{-1} = \mathbf{I} + \lambda\mathbf{K}$.

Some vector calculus shows that $\hat{\mathbf{Y}}_\lambda = \mathbf{S}_\lambda \mathbf{Y}$ is a solution to the minimization problem

$$\underset{\mathbf{v} \in \mathbb{R}^N}{\text{minimize}} ((\mathbf{Y} - \mathbf{v})' (\mathbf{Y} - \mathbf{v}) + \lambda \mathbf{v}' \mathbf{K} \mathbf{v}) \tag{81}$$

so that this matrix \mathbf{K} can be thought of as defining a "penalty" in fitting a smoothed version of \mathbf{Y} .

Then, since \mathbf{S}_λ is symmetric non-negative definite, it has an eigen decomposition as

$$\mathbf{S}_\lambda = \mathbf{U}\mathbf{D}\mathbf{U}' = \sum_{j=1}^N d_j \mathbf{u}_j \mathbf{u}_j' \quad (82)$$

where columns of \mathbf{U} (the eigenvectors \mathbf{u}_j) comprise an orthonormal basis for \Re^N and

$$\mathbf{D} = \mathbf{diag}(d_1, d_2, \dots, d_N)$$

for eigenvalues of \mathbf{S}_λ

$$d_1 \geq d_2 \geq \dots \geq d_N > 0$$

It turns out to be guaranteed that $d_1 = d_2 = 1$.

Consider how the eigenvalues and eigenvectors of \mathbf{S}_λ are related to those for \mathbf{K} . An eigenvalue for \mathbf{K} , say η , solves

$$\det(\mathbf{K} - \eta\mathbf{I}) = 0$$

Now

$$\det(\mathbf{K} - \eta\mathbf{I}) = \det\left(\frac{1}{\lambda}[(\mathbf{I} + \lambda\mathbf{K}) - (1 + \lambda\eta)\mathbf{I}]\right)$$

So $1 + \lambda\eta$ must be an eigenvalue of $\mathbf{I} + \lambda\mathbf{K}$ and $1/(1 + \lambda\eta)$ must be an eigenvalue of $\mathbf{S}_\lambda = (\mathbf{I} + \lambda\mathbf{K})^{-1}$. So for some j we must have

$$d_j = \frac{1}{1 + \lambda\eta}$$

and observing that $1/(1 + \lambda\eta)$ is decreasing in η , we may conclude that

$$d_j = \frac{1}{1 + \lambda\eta_{N-j+1}} \quad (83)$$

for

$$\eta_1 \geq \eta_2 \geq \dots \geq \eta_{N-2} \geq \eta_{N-1} = \eta_N = 0$$

the eigenvalues of \mathbf{K} (that themselves *do not* depend upon λ). So, for example, in light of facts (79), (82), and (83), the smoothing effective degrees of freedom are

$$\text{df}(\lambda) = \text{tr}(\mathbf{S}_\lambda) = \sum_{j=1}^N d_j = 2 + \sum_{j=1}^{N-2} \frac{1}{1 + \lambda\eta_j}$$

which is clearly decreasing in λ (with minimum value 2 in light of the fact that \mathbf{S}_λ has two eigenvalues that are 1).

Further, consider \mathbf{u}_j , the eigenvector of \mathbf{S}_λ corresponding to eigenvalue d_j . $\mathbf{S}_\lambda \mathbf{u}_j = d_j \mathbf{u}_j$ so that

$$\mathbf{u}_j = \mathbf{S}_\lambda^{-1} d_j \mathbf{u}_j = (\mathbf{I} + \lambda\mathbf{K}) d_j \mathbf{u}_j$$

so that

$$\mathbf{u}_j = d_j \mathbf{u}_j + d_j \lambda \mathbf{K} \mathbf{u}_j$$

and thus

$$\mathbf{K} \mathbf{u}_j = \left(\frac{1 - d_j}{\lambda d_j} \right) \mathbf{u}_j = \eta_{N-j+1} \mathbf{u}_j$$

That is, \mathbf{u}_j is an eigenvector of \mathbf{K} corresponding to the $(N - j + 1)$ st largest eigenvalue. That is, for all λ the eigenvectors of \mathbf{S}_λ are eigenvectors of \mathbf{K} and *thus do not depend upon λ* .

Then, for any λ

$$\begin{aligned} \widehat{\mathbf{Y}}_\lambda &= \mathbf{S}_\lambda \mathbf{Y} = \left(\sum_{j=1}^N d_j \mathbf{u}_j \mathbf{u}_j' \right) \mathbf{Y} \\ &= \sum_{j=1}^N d_j \langle \mathbf{u}_j, \mathbf{Y} \rangle \mathbf{u}_j \\ &= \langle \mathbf{u}_1, \mathbf{Y} \rangle \mathbf{u}_1 + \langle \mathbf{u}_2, \mathbf{Y} \rangle \mathbf{u}_2 + \sum_{j=3}^N \frac{\langle \mathbf{u}_j, \mathbf{Y} \rangle}{1 + \lambda \eta_{N-j+1}} \mathbf{u}_j \end{aligned} \quad (84)$$

and we see that $\widehat{\mathbf{Y}}_\lambda$ is a shrunken version of \mathbf{Y} (that progresses from \mathbf{Y} to the projection of \mathbf{Y} onto the span of $\{\mathbf{u}_1, \mathbf{u}_2\}$ as λ runs from 0 to ∞)²¹. The larger is λ , the more severe the shrinking overall. Further, the larger is j , the smaller is d_j and the more severe is the shrinking in the \mathbf{u}_j direction. (The unpenalized directions \mathbf{u}_1 and \mathbf{u}_2 have no associated shrinking.) In the context of cubic smoothing splines, large j correspond to "wiggly" (as a functions of coordinate i or value of the input x_i) \mathbf{u}_j , and the prescription (84) calls for suppression of "wiggly" components of \mathbf{Y} .

Further, since $\widehat{\mathbf{Y}}_\lambda = \mathbf{H} \widehat{\boldsymbol{\beta}}_\lambda$ and \mathbf{H} is nonsingular, as λ runs from 0 to ∞ , $\widehat{\boldsymbol{\beta}}_\lambda$ runs from $\mathbf{H}^{-1} \mathbf{Y}$ to $\mathbf{H}^{-1} (\langle \mathbf{u}_1, \mathbf{Y} \rangle \mathbf{u}_1 + \langle \mathbf{u}_2, \mathbf{Y} \rangle \mathbf{u}_2)$. And there is "shrinking" enforced on $\widehat{\boldsymbol{\beta}}_\lambda$ in the sense that the quadratic form $\widehat{\boldsymbol{\beta}}_\lambda' \boldsymbol{\Omega} \widehat{\boldsymbol{\beta}}_\lambda$ must be non-increasing in λ . (If not, the fact that $\|\mathbf{Y} - \widehat{\mathbf{Y}}_\lambda\|^2$ increases in λ would produce a contradiction.)

Notice that large j correspond to *early/large* eigenvalues of the penalty matrix \mathbf{K} in (81). Letting $\mathbf{u}_j^* = \mathbf{u}_{N-j+1}$ so that

$$\begin{aligned} \mathbf{U}^* &= (\mathbf{u}_N, \mathbf{u}_{N-1}, \dots, \mathbf{u}_1) \\ &= (\mathbf{u}_1^*, \mathbf{u}_2^*, \dots, \mathbf{u}_N^*) \end{aligned}$$

the eigen decomposition of \mathbf{K} is

$$\mathbf{K} = \mathbf{U}^* \mathit{diag}(\eta_1, \eta_2, \dots, \eta_N) \mathbf{U}^{*t}$$

²¹It is possible to argue that the span of $\{\mathbf{u}_1, \mathbf{u}_2\}$ is the set of vectors of the form $c\mathbf{1} + d\mathbf{x}$, as is consistent with the integral penalty in original function optimization problem.

and criterion (81) can be written as

$$\underset{\mathbf{v} \in \mathfrak{R}^N}{\text{minimize}} \left((\mathbf{Y} - \mathbf{v})' (\mathbf{Y} - \mathbf{v}) + \lambda \mathbf{v}' \mathbf{U}^* \mathbf{diag}(\eta_1, \eta_2, \dots, \eta_N) \mathbf{U}^{*'} \mathbf{v} \right)$$

or equivalently as

$$\underset{\mathbf{v} \in \mathfrak{R}^N}{\text{minimize}} \left((\mathbf{Y} - \mathbf{v})' (\mathbf{Y} - \mathbf{v}) + \lambda \sum_{j=1}^{N-2} \eta_j \langle \mathbf{u}_j^*, \mathbf{v} \rangle^2 \right) \quad (85)$$

(since $\eta_{N-1} = \eta_N = 0$) and we see that eigenvalues of \mathbf{K} function as penalty coefficients applied to the N orthogonal components of $\mathbf{v} = \sum_{j=1}^N \langle \mathbf{u}_j^*, \mathbf{v} \rangle \mathbf{u}_j^*$ in the choice of optimizing \mathbf{v} . From this point of view, the \mathbf{u}_j (or \mathbf{u}_j^*) provide the natural alternative (to the columns of \mathbf{H}) basis (for \mathfrak{R}^N) for representing or approximating \mathbf{Y} , and the last equality in display (84) provides an explicit form for the optimizing smoothed vector $\hat{\mathbf{Y}}_\lambda$.

In this development, \mathbf{K} has had a specific meaning derived from the \mathbf{H} and $\mathbf{\Omega}$ matrices connected specifically with smoothing splines *and* the particular values of x in the training dataset. But in the end, an interesting possibility brought up by the whole development is that of forgetting the origins (from \mathbf{K}) of the η_j and \mathbf{u}_j and beginning with any interesting/intuitively appealing orthonormal basis $\{\mathbf{u}_j\}$ and set of non-negative penalties $\{\eta_j\}$ for use in minimization (85). Working backwards through relationships (84) and (83) one is then led to the corresponding smoothed vector $\hat{\mathbf{Y}}_\lambda$ and smoothing matrix \mathbf{S}_λ . (More detail on this matter is in Section 5.3.)

It is also worth remarking that since $\hat{\mathbf{Y}}_\lambda = \mathbf{S}_\lambda \mathbf{Y}$ the rows of \mathbf{S}_λ provide weights to be applied to the elements of \mathbf{Y} in order to produce predictions/smoothed values corresponding to \mathbf{Y} . These can for each i be thought of as defining a corresponding "equivalent kernel" (for an appropriate "kernel-weighted average" of the training output values as discussed in Section 6.1). (See Figure 5.8 of HTF2 in this regard.)

5.2 Multi-Dimensional Smoothing Splines

If $p = 2$ and the vector of inputs, \mathbf{x} , takes values in \mathfrak{R}^2 , one might propose to seek

$$\hat{f}_\lambda = \underset{\text{functions } h \text{ with 2 derivatives}}{\text{arg min}} \left(\sum_{i=1}^N (y_i - h(\mathbf{x}_i))^2 + \lambda J[h] \right)$$

for

$$J[h] \equiv \iint_{\mathfrak{R}^2} \left(\frac{\partial^2 h}{\partial x_1^2} \right)^2 + 2 \left(\frac{\partial^2 h}{\partial x_1 \partial x_2} \right)^2 + \left(\frac{\partial^2 h}{\partial x_2^2} \right)^2 dx_1 dx_2$$

An optimizing $\hat{f}_\lambda : \mathfrak{R}^2 \rightarrow \mathfrak{R}$ can be identified and is called a "thin plate spline." As $\lambda \rightarrow 0$, \hat{f}_λ becomes an interpolator, as $\lambda \rightarrow \infty$ it defines the OLS plane

through the data in 3-space. In general, it can be shown to be of the form

$$f_\lambda(\mathbf{x}) = \beta_{0\lambda} + \beta'_\lambda \mathbf{x} + \sum_{i=1}^N \alpha_{i\lambda} g_i(\mathbf{x}) \quad (86)$$

where $g_i(\mathbf{x}) = \eta(\|\mathbf{x} - \mathbf{x}_i\|)$ for $\eta(z) = z^2 \ln z^2$. The $g_i(\mathbf{x})$ are "radial basis functions" (radially symmetric basis functions) and fitting is accomplished much as for the $p = 1$ case. The form (86) is plugged into the optimization criterion and a discrete penalized least squares problem emerges (after taking account of some linear constraints that are required to keep $J[f_\lambda] < \infty$). HTF seem to indicate that in order to keep computations from exploding with N , it usually suffices to replace the N functions $g_i(\mathbf{x})$ in form (86) with $K \ll N$ functions $g_i^*(\mathbf{x}) = \eta(\|\mathbf{x} - \mathbf{x}_i^*\|)$ for K potential input vectors \mathbf{x}_i^* placed on a rectangular grid covering the convex hull of the N training data input vectors \mathbf{x}_i .

For large p , one might simply declare that attention is going to be limited to predictors of some restricted form, and for h in that restricted class, seek to optimize

$$\sum_{i=1}^N (y_i - h(\mathbf{x}_i))^2 + \lambda J[h]$$

for $J[h]$ some appropriate penalty on h intended to regularize/restrict its wiggling. For example, one might assume that a form

$$g(\mathbf{x}) = \sum_{j=1}^p g_j(x_j)$$

will be used and set

$$J[g] = \sum_{j=1}^p \int (g_j''(x))^2 dx$$

and be led to additive splines.

Or, one might assume that

$$g(\mathbf{x}) = \sum_{j=1}^p g_j(x_j) + \sum_{j,k} g_{jk}(x_j, x_k) \quad (87)$$

and invent an appropriate penalty function. It seems like a sum of 1-d smoothing spline penalties on the g_j and 2-d thin plate spline penalties on the g_{jk} is the most obvious starting point. Details of fitting are a bit murky (though I am sure that they can be found in book on generalized additive models). Presumably one cycles through the summands in display (87) iteratively fitting functions to sets of residuals defined by the original y_i minus the sums of all other current versions of the components until some convergence criterion is satisfied. Function (87) has a kind of "main effects plus 2-factor interactions" form, but it is (at least in theory) possible to also consider higher order terms in this kind of expansion.

5.3 An Abstraction of the Smoothing Spline Material and Penalized Fitting in \mathfrak{R}^N

In abstraction of the smoothing spline development, suppose that $\{\mathbf{u}_j\}$ is a set of $M \leq N$ orthonormal N -vectors, $\lambda \geq 0, \eta_j \geq 0$ for $j = 1, 2, \dots, M$, and consider the optimization problem

$$\text{minimize}_{\mathbf{v} \in \text{span}\{\mathbf{u}_j\}} \left((\mathbf{Y} - \mathbf{v})' (\mathbf{Y} - \mathbf{v}) + \lambda \sum_{j=1}^M \eta_j \langle \mathbf{u}_j, \mathbf{v} \rangle^2 \right)$$

For $\mathbf{v} = \sum_{j=1}^M c_j \mathbf{u}_j \in \text{span}\{\mathbf{u}_j\}$, the penalty is $\lambda \sum_{j=1}^M \eta_j \langle \mathbf{u}_j, \mathbf{v} \rangle^2 = \lambda \sum_{j=1}^M \eta_j c_j^2$ and in this penalty, $\lambda \eta_j$ is a multiplier of the squared length of the component of \mathbf{v} in the direction of \mathbf{u}_j . The optimization criterion is then

$$(\mathbf{Y} - \mathbf{v})' (\mathbf{Y} - \mathbf{v}) + \lambda \sum_{j=1}^M \eta_j \langle \mathbf{u}_j, \mathbf{v} \rangle^2 = \sum_{j=1}^M (\langle \mathbf{u}_j, \mathbf{Y} \rangle - c_j)^2 + \lambda \sum_{j=1}^M \eta_j c_j^2$$

and it is then easy to see (via simple calculus) that

$$c_j^{\text{opt}} = \frac{\langle \mathbf{u}_j, \mathbf{Y} \rangle}{1 + \lambda \eta_j}$$

i.e.

$$\hat{\mathbf{Y}} = \mathbf{v}^{\text{opt}} = \sum_{j=1}^M \frac{\langle \mathbf{u}_j, \mathbf{Y} \rangle}{1 + \lambda \eta_j} \mathbf{u}_j$$

From this it's clear how the penalty structure dictates optimally shrinking the components of the projection of \mathbf{Y} onto $\text{span}\{\mathbf{u}_j\}$.

It is further worth noting that for a given set of penalty coefficients, $\hat{\mathbf{Y}}$ can be represented as $\mathbf{S}\mathbf{Y}$ for

$$\mathbf{S} = \sum_{j=1}^M d_j \mathbf{u}_j \mathbf{u}_j' = \mathbf{U} \text{diag} \left(\frac{1}{1 + \lambda \eta_1}, \dots, \frac{1}{1 + \lambda \eta_M} \right) \mathbf{U}'$$

for $\mathbf{U} = (\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_M)$. Then it's easy to see that smoother matrix \mathbf{S} is a rank M matrix for which $\hat{\mathbf{Y}} = \mathbf{S}\mathbf{Y}$.

One context in which this material might find immediate application is where some set of basis functions $\{h_j\}$ are increasingly "wiggly" with increasing j and the vectors \mathbf{u}_j come from applying the Gram-Schmidt process to the vectors

$$\mathbf{h}_j = (h_j(\mathbf{x}_1), \dots, h_j(\mathbf{x}_N))'$$

In this context, it would be very natural to penalize the later \mathbf{u}_j more severely than the early ones.

5.4 Graph-Based Penalized Fitting/Smoothing (and Semi-Supervised Learning)

Another interesting smoothing methodology related to the material of the three previous sections concerns use of fitting penalties based on the graph Laplacians introduced in Section 2.4.3.²² Consider then N complete data cases $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)$ and $M \geq 0$ additional data cases where only inputs $\mathbf{x}_{N+1}, \dots, \mathbf{x}_{N+M}$ are available. There is no necessity here that $M > 0$, but it can be so in the event that predictions are desired at $\mathbf{x}_{N+1}, \dots, \mathbf{x}_{N+M}$ whose values might not be in the training set. Where there are $M > 0$ genuine "unlabeled cases" whose inputs are assumed to come from the same mechanism as the inputs $\mathbf{x}_1, \dots, \mathbf{x}_N$ and might be used to more or less "fill in" the relevant part of the input space not covered by the complete/labeled data cases, the terminology **semi-supervised learning** is sometimes used to describe the building of a predictor for y at all $N + M$ input vectors. The case $M = 1$ might be used to simply make a single prediction at a single input not exactly seen in a "usual" training set of N complete data pairs.

Suppose that following the development of Section 2.4.3 one can make an adjacency matrix based on the $N + M$ input vectors,

$$\mathbf{S} = (s_{ij})_{\substack{i=1, \dots, N+M \\ j=1, \dots, N+M}} = \begin{pmatrix} \mathbf{S}_L & \mathbf{S}_{LU} \\ \mathbf{S}_{UL} & \mathbf{S}_U \end{pmatrix}$$

and corresponding Laplacian and symmetric normalized Laplacian, respectively

$$\mathbf{L} = \begin{pmatrix} \mathbf{L}_L & \mathbf{L}_{LU} \\ \mathbf{L}_{UL} & \mathbf{L}_U \end{pmatrix} \text{ and } \mathbf{L}^* = \begin{pmatrix} \mathbf{L}_L^* & \mathbf{L}_{LU}^* \\ \mathbf{L}_{UL}^* & \mathbf{L}_U^* \end{pmatrix}$$

Then with

$$\mathbf{Y}_{(N+M) \times 1} = \begin{pmatrix} \mathbf{Y}_L \\ \mathbf{Y}_U \end{pmatrix}$$

what one might wish to do is produce a vector of smoothed/fitted values $\hat{\mathbf{Y}}_{(N+M) \times 1}$ such that entries corresponding to input vectors with large adjacencies tend to be alike. This is possible in way highly reminiscent of the material in Sections 5.1 and 5.3.

For $\mathbf{v} \in \mathfrak{R}^{N+M}$ written as

$$\mathbf{v}_{(N+M) \times 1} = \begin{pmatrix} \mathbf{v}_L \\ \mathbf{v}_U \end{pmatrix}$$

²²The material here is adapted from "Graph-Based Semi-Supervised Learning with BIG Data" by Banerjee, Culp, Ryan, and Michailidis, that appeared in *Research on Applied Cybernetics and System Science* in 2017.

consider the optimization problem in \Re^{N+M}

$$\underset{\mathbf{v} \in \Re^{N+M}}{\text{minimize}} \left((\mathbf{Y}_L - \mathbf{v}_L)' (\mathbf{Y}_L - \mathbf{v}_L) + \lambda \mathbf{v}' \mathbf{L} \mathbf{v} \right) \quad (88)$$

for some $\lambda > 0$ (or the same with \mathbf{L}^* replacing \mathbf{L} in the quadratic penalty term). The developments (52) and (53) of Section 2.4.3 show that upon expanding \mathbf{v} in terms of the $N + M$ (orthonormal) eigenvectors of \mathbf{L} (or \mathbf{L}^*) it follows that components of \mathbf{v} that are multiples of late eigenvectors (ones with small eigenvalues)

1. have similar entries for cases with large adjacencies, and
2. are relatively lightly penalized in the minimization.

This strongly suggests that solutions to the optimization problem (88) will provide smoothed prediction vectors $\hat{\mathbf{Y}}$ where entries with corresponding inputs with large adjacencies are similar.

Recent work of Culp and Ryan provides theory, methods, and software for solving the problem (88) and many nice generalizations of it (including consideration of losses other than SEL that produce methods for classification problems). For purposes of exposition here, we will provide the explicit solution that is available for the SEL problem. It turns out that the problem (88) and generalizations of it separate nicely into two parts. That is

$$\hat{\mathbf{Y}}_U^{\text{opt}} = -\mathbf{L}_U^{-1} \mathbf{L}_{UL} \hat{\mathbf{Y}}_L^{\text{opt}} \quad (89)$$

(or the same with \mathbf{L}^* s replacing \mathbf{L} s) where $\hat{\mathbf{Y}}_L^{\text{opt}} = \mathbf{v}_L$ solving

$$\underset{\mathbf{v}_L \in \Re^N}{\text{minimize}} \left((\mathbf{Y}_L - \mathbf{v}_L)' (\mathbf{Y}_L - \mathbf{v}_L) + \lambda \mathbf{v}_L' \tilde{\mathbf{L}}_L \mathbf{v}_L \right) \quad (90)$$

for $\tilde{\mathbf{L}}_L = \mathbf{L}_L - \mathbf{L}_{LU} \mathbf{L}_U^{-1} \mathbf{L}_{UL}$ (or, again, the same with \mathbf{L}^* s replacing \mathbf{L} s). (Generalizations of the development here replace SSE in displays (88) and (90) with other losses, but the form (89) is unchanged.) But the problem (90) is familiar and its solution a simple consequence of vector calculus

$$\hat{\mathbf{Y}}_L^{\text{opt}} = \left(\mathbf{I} + \lambda \tilde{\mathbf{L}}_L \right)^{-1} \mathbf{Y}_L$$

This is exactly parallel to the displays (80) and (81) and the discussion around them. $\left(\mathbf{I} + \lambda \tilde{\mathbf{L}}_L \right)^{-1}$ (and its starred version) is a smoother/shrinker matrix. Further, the matrix $-\mathbf{L}_U^{-1} \mathbf{L}_{UL}$ in display (89) and its starred version are stochastic matrices and entries of $\hat{\mathbf{Y}}_U^{\text{opt}}$ are averages of the elements of $\hat{\mathbf{Y}}_L^{\text{opt}}$.

6 Kernel and Local Regression Smoothing Methods and SEL Prediction

The central idea of this material is that when finding $\hat{f}(x_0)$ one might weight points in the training set according to how close they are to x_0 , do some kind of fitting around x_0 , and ultimately read off the value of the fit at x_0 .

6.1 One-dimensional Kernel and Local Regression Smoothers

For the time being, suppose that x takes values in $[0, 1]$. Invent weighting schemes for points in the training set by defining a (usually, symmetric about 0) non-negative, real-valued function $D(t)$ that is non-increasing for $t \geq 0$ and non-decreasing for $t \leq 0$. Often $D(t)$ is taken to have value 0 unless $|t| \leq 1$. Then, a kernel function²³ is

$$\mathcal{K}_\lambda(x, x_0) = D\left(\frac{x - x_0}{\lambda}\right) \quad (91)$$

where λ is a "bandwidth" parameter that controls the rate at which weights drop off as one moves away from x_0 (and indeed in the case that $D(t) = 0$ for $|t| > 1$, how far one moves away from x_0 before no weight is assigned). Common choices for D are

1. the Epanechnikov quadratic kernel, $D(t) = \frac{3}{4}(1 - t^2)I[|t| \leq 1]$,
2. the "tri-cube" kernel, $D(t) = (1 - |t|^3)^3 I[|t| \leq 1]$, and
3. the standard normal density, $D(t) = \phi(t)$.

These three are pictured in Figure 24.

Using weights (91) to make a weighted average of training responses, one arrives at the Nadaraya-Watson kernel-weighted prediction at x_0

$$\hat{f}_\lambda(x_0) = \frac{\sum_{i=1}^N \mathcal{K}_\lambda(x_0, x_i) y_i}{\sum_{i=1}^N \mathcal{K}_\lambda(x_0, x_i)} \quad (92)$$

This typically smooths training outputs y_i in a more pleasing way than does a k -nearest neighbor average, but it has obvious problems at the ends of the interval $[0, 1]$ and at places in the interior of the interval where training data are dense to one side of x_0 and sparse to the other, if the target $E[y|x = z]$ has non-zero derivative at $z = x_0$. For example, at $x_0 = 1$ only $x_i \leq 1$ get weight, and if $E[y|x = z]$ is decreasing at $z = x_0 = 1$, $\hat{f}_\lambda(1)$ will be positively biased. That is, with usual symmetric kernels, predictor (92) will fail to adequately follow an obvious trend at 0 or 1 (or at any point between where there is a sharp change in the density of input values in the training set).

²³This is again a potentially different usage of the word "kernel" than that in Section 1.4.3 and no non-negative definiteness of the function is needed or assumed.

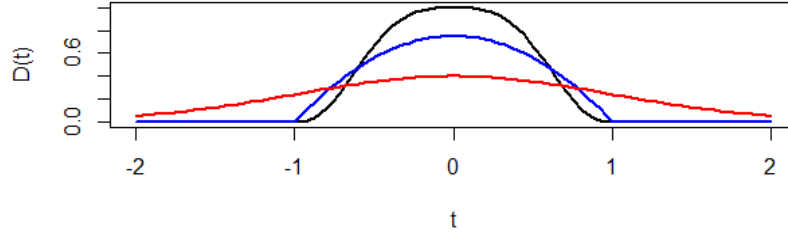


Figure 24: Three standard choices of $D(t)$: Epanechnikov quadratic kernel (blue), tricube (black), and standard normal density (red).

A way to address this problem with the Nadaraya-Watson predictor is to replace the locally-fitted constant with a locally-fitted line. That is, at x_0 one might choose $\alpha(x_0)$ and $\beta(x_0)$ to solve the optimization problem

$$\underset{\alpha \text{ and } \beta}{\text{minimize}} \sum_{i=1}^N \mathcal{K}_\lambda(x_0, x_i) (y_i - (\alpha + \beta x_i))^2 \quad (93)$$

and then employ the prediction

$$\hat{f}_\lambda(x_0) = \alpha(x_0) + \beta(x_0)x_0 \quad (94)$$

Now the weighted least squares problem (93) has an explicit solution. Let

$$\mathbf{B}_{N \times 2} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{pmatrix}$$

and take

$$\mathbf{W}_{N \times N}(x_0) = \mathbf{diag}(\mathcal{K}_\lambda(x_0, x_1), \dots, \mathcal{K}_\lambda(x_0, x_N))$$

then predictor (94) is

$$\begin{aligned} \hat{f}_\lambda(x_0) &= (1, x_0) (\mathbf{B}' \mathbf{W}(x_0) \mathbf{B})^{-1} \mathbf{B}' \mathbf{W}(x_0) \mathbf{Y} \\ &= \mathbf{l}'(x_0) \mathbf{Y} \end{aligned} \quad (95)$$

for the $1 \times N$ vector $\mathbf{l}'(x_0) = (1, x_0) (\mathbf{B}' \mathbf{W}(x_0) \mathbf{B})^{-1} \mathbf{B}' \mathbf{W}(x_0)$. It is thus obvious that locally weighted linear regression is (an albeit x_0 -dependent) *linear* operation on the vector of outputs. The weights in $\mathbf{l}'(x_0)$ combine the

original kernel values and the least squares fitting operation to produce a kind of "equivalent kernel" (for a Nadaraya-Watson type weighted average).

Recall that for smoothing splines, smoothed values are

$$\widehat{\mathbf{Y}}_\lambda = \mathbf{S}_\lambda \mathbf{Y}$$

where the parameter λ is the penalty weight, and

$$\text{df}(\lambda) = \text{tr}(\mathbf{S}_\lambda)$$

We may do something parallel in the present context. We may take

$$\mathbf{L}_\lambda = \begin{pmatrix} \mathbf{l}'(x_1) \\ \mathbf{l}'(x_2) \\ \vdots \\ \mathbf{l}'(x_N) \end{pmatrix}$$

where now the parameter λ is the bandwidth, write

$$\widehat{\mathbf{Y}}_\lambda = \mathbf{L}_\lambda \mathbf{Y}$$

and define

$$\text{df}(\lambda) = \text{tr}(\mathbf{L}_\lambda)$$

HTF suggest that matching degrees of freedom for a smoothing spline and a kernel smoother produces very similar equivalent kernels, smoothers, and predictions.

There is a famous theorem of Silverman that adds technical credence to this notion. Roughly the theorem says that for large N , if in the case $p = 1$ the inputs x_1, x_2, \dots, x_N are iid with density $p(x)$ on $[a, b]$, λ is neither too big nor too small,

$$D_S(u) = \frac{1}{2} \exp\left(-\frac{|u|}{\sqrt{2}}\right) \sin\left(\frac{|u|}{\sqrt{2}} + \frac{\pi}{4}\right)$$

$$\gamma(x) = \left(\frac{\lambda}{Np(x)}\right)^{1/4}$$

and

$$G_\lambda(z, x) = \frac{1}{\gamma(x)p(x)} D_S\left(\frac{z-x}{\gamma(x)}\right)$$

then for x_i not too close to either a or b ,

$$(\mathbf{S}_\lambda)_{ij} \approx \frac{1}{N} G_\lambda(x_i, x_j)$$

(in some appropriate probabilistic sense) and the smoother matrix for cubic spline smoothing has entries like those that would come from an appropriate kernel smoothing.

6.2 Local Regression Smoothing in p Dimensions

A direct generalization of 1-dimensional local regression smoothing to p dimensions might go roughly as follows. For D as before, and $\mathbf{x} \in \mathfrak{R}^p$, one might set

$$\mathcal{K}_\lambda(\mathbf{x}_0, \mathbf{x}) = D \left(\frac{\|\mathbf{x} - \mathbf{x}_0\|}{\lambda} \right) \quad (96)$$

and fit linear forms locally by choosing $\alpha(\mathbf{x}_0) \in \mathfrak{R}$ and $\beta(\mathbf{x}_0) \in \mathfrak{R}^p$ to solve the optimization problem

$$\underset{\alpha \text{ and } \beta}{\text{minimize}} \sum_{i=1}^N \mathcal{K}_\lambda(\mathbf{x}_0, \mathbf{x}_i) (y_i - (\alpha + \beta' \mathbf{x}_i))^2$$

and predicting as

$$\hat{f}_\lambda(\mathbf{x}_0) = \alpha(\mathbf{x}_0) + \beta'(\mathbf{x}_0) \mathbf{x}_0$$

This seems typically to be done only after standardizing the coordinates of \mathbf{x} and can be effective as long as N is not too small and p is not more than 2 or 3. However for $p > 3$, the curse of dimensionality comes into play and N points usually just aren't dense enough in p -space to make direct use of kernel smoothing effective. If the method is going to be successful in \mathfrak{R}^p it will need to be applied under appropriate structure assumptions.

One way to apply additional structure to the p -dimensional kernel smoothing problem is to essentially reduce input variable dimension by replacing the kernel (96) with the "structured kernel"

$$\mathcal{K}_{\lambda, \mathbf{A}}(\mathbf{x}_0, \mathbf{x}) = D \left(\frac{\sqrt{(\mathbf{x} - \mathbf{x}_0)' \mathbf{A} (\mathbf{x} - \mathbf{x}_0)}}{\lambda} \right)$$

for an appropriate non-negative definite matrix \mathbf{A} . For the eigen decomposition of \mathbf{A} ,

$$\mathbf{A} = \mathbf{V} \mathbf{D} \mathbf{V}'$$

write

$$(\mathbf{x} - \mathbf{x}_0)' \mathbf{A} (\mathbf{x} - \mathbf{x}_0) = \left(\mathbf{D}^{\frac{1}{2}} \mathbf{V}' (\mathbf{x} - \mathbf{x}_0) \right)' \left(\mathbf{D}^{\frac{1}{2}} \mathbf{V}' (\mathbf{x} - \mathbf{x}_0) \right)$$

This amounts to using not \mathbf{x} and \mathfrak{R}^p distance from \mathbf{x} to \mathbf{x}_0 to define weights, but rather $\mathbf{D}^{\frac{1}{2}} \mathbf{V}' \mathbf{x}$ and \mathfrak{R}^p distance from $\mathbf{D}^{\frac{1}{2}} \mathbf{V}' \mathbf{x}$ to $\mathbf{D}^{\frac{1}{2}} \mathbf{V}' \mathbf{x}_0$. In the event that some entries of \mathbf{D} are 0 (or are nearly so), this basically reduces dimension from p to the number of large eigenvalues of \mathbf{A} and defines weights in a space of that dimension (spanned by eigenvectors corresponding to non-zero eigenvalues) where the curse of dimensionality may *not* preclude effective use of kernel smoothing. The "trick" is, of course, identifying the right directions into which to project. (Searching for such directions is part of the Friedman "projection pursuit" ideas discussed below.)

7 High-Dimensional Use of Low-Dimensional Smoothers and SEL Prediction

There are several ways that have been suggested for making use of fairly low-dimensional (and thus, potentially effective) smoothing in large p problems. One of them is the "structured kernels" idea just discussed. Two more follow.

7.1 Structured Regression Functions

7.1.1 Additive Models

A way to apply structure to the p -dimensional smoothing problem is through assumptions on the form of the predictor fit. For example, one might assume additivity in a form

$$f(\mathbf{x}) = \alpha + \sum_{j=1}^p g_j(x_j) \quad (97)$$

and try to do fitting of the p functions g_j and constant α .

One more or less ad hoc method of fitting forms like form (97) is the so-called "**back-fitting algorithm**." That is to (generalize form (97) slightly and) fit (under SEL)

$$f(\mathbf{x}) = \alpha + \sum_{l=1}^L g_l(\mathbf{x}^l) \quad (98)$$

for \mathbf{x}^l some part of \mathbf{x} , one might set $\hat{\alpha} = \frac{1}{N} \sum_{i=1}^N y_i$, and then cycle through $l = 1, 2, \dots, L, 1, 2, \dots$

1. fitting via some appropriate (often linear) operation (e.g., spline or kernel smoothing)

$$g_l(\mathbf{x}^l) \text{ to "data" } \{(\mathbf{x}_i^l, y_i^l)\}_{i=1,2,\dots,N}$$

for

$$y_i^l = y_i - \left(\hat{\alpha} + \sum_{m \neq l} g_m(\mathbf{x}_i^m) \right)$$

where the g_m are the current versions of the fitted summands,

2. setting

g_l = the newly fitted version

– the sample mean of this newly fitted version across all \mathbf{x}_i^l

(in theory this is not necessary, but it is here to prevent numerical/round-off errors from causing the g_m to drift up and down by additive constants summing to 0 across m),

3. iterating until convergence to, say, $\hat{f}(\mathbf{x})$.

A more principled SEL fitting methodology for additive forms like that in display (98) (e.g. implemented by Wood in his `mgcf` R package) is the simultaneous fitting of α and all the functions g_l via penalized least squares. That is, using an appropriate set of basis functions for smooth functions of \mathbf{x}^l (often a tensor product basis in the event that the dimension of \mathbf{x}^l is more than 1) each g_l might be represented as a linear combination of those basis functions. Then form (98) is in fact a constant plus a linear combination of basis functions. So upon adopting a quadratic penalty for the coefficients, one has a kind of ridge regression problem and explicit forms for all fitted coefficients and $\hat{\alpha}$. The practical details of making the various bases and picking ridge parameters, etc. are not trivial, but the basic idea is clear.

The simplest version of this line of development, based on form (97), might be termed fitting of a "main effects model." But the approach might as well be applied to fit a "main effects and two factor interactions model," using some g_l s that are functions of only one coordinate of \mathbf{x} and others that depend upon only two coordinates of the input vector. One may mix types of predictors (continuous, categorical) and types of functions of them in the additive form to produce all sorts of interesting models (including semi-parametric ones and ones with low order interactions).

7.1.2 Other Structured Regression Forms

Another possibility for introducing structure assumptions and making use of low-dimensional smoothing in a large p situation, is by making strong global assumptions on the forms of the influence of some input variables on the output, but allowing parameters of those forms to vary in a flexible fashion with the values of some small number of coordinates of \mathbf{x} . For sake of example, suppose that $p = 4$. One might consider predictor forms

$$f(\mathbf{x}) = \alpha(x_3, x_4) + \beta_1(x_3, x_4)x_1 + \beta_2(x_3, x_4)x_2$$

That is, one might assume that for fixed (x_3, x_4) , the form of the predictor is linear in (x_1, x_2) , but that the coefficients of that form may change in a flexible way with (x_3, x_4) . Fitting might then be approached by locally weighted least squares, *with only (x_3, x_4) involved in the setting of the weights*. That is, one might for each (x_{30}, x_{40}) , minimize over choices of $\alpha(x_{30}, x_{40})$, $\beta_1(x_{30}, x_{40})$ and $\beta_2(x_{30}, x_{40})$ the weighted sum of squares

$$\sum_{i=1}^N \mathcal{K}_\lambda((x_{30}, x_{40}), (x_{3i}, x_{4i})) (y_i - (\alpha(x_{30}, x_{40}) + \beta_1(x_{30}, x_{40})x_{1i} + \beta_2(x_{30}, x_{40})x_{2i}))^2$$

and then employ the predictor

$$\hat{f}(\mathbf{x}_0) = \hat{\alpha}(x_{30}, x_{40}) + \hat{\beta}_1(x_{30}, x_{40})x_{10} + \hat{\beta}_2(x_{30}, x_{40})x_{20}$$

This kind of device keeps the dimension of the space where one is doing smoothing down to something manageable. But note that nothing here does any thresholding or automatic variable selection.

7.2 Projection Pursuit Regression

For $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_M$ unit p -vectors of parameters, we might consider as predictors fitted versions of the form

$$f(\mathbf{x}) = \sum_{m=1}^M g_m(\mathbf{w}'_m \mathbf{x}) \quad (99)$$

This is an additive form in the derived variables $v_m = \mathbf{w}'_m \mathbf{x}$. The functions g_m and the directions \mathbf{w}_m are to be fit from the training data. The $M = 1$ case of this form is the "single index model" of econometrics.

How does one fit a predictor of this form (99)? Consider first the $M = 1$ case. Given \mathbf{w} , there are pairs (v_i, y_i) for $v_i = \mathbf{w}' \mathbf{x}_i$ and a 1-dimensional smoothing method can be used to estimate g . On the other hand, given g , one might seek to optimize \mathbf{w} via an iterative search. A Gauss-Newton algorithm can be based on the first order Taylor approximation

$$g(\mathbf{w}' \mathbf{x}_i) \approx g(\mathbf{w}'_{\text{old}} \mathbf{x}_i) + g'(\mathbf{w}'_{\text{old}} \mathbf{x}_i) (\mathbf{w}' - \mathbf{w}'_{\text{old}}) \mathbf{x}_i$$

so that

$$\sum_{i=1}^N (y_i - g(\mathbf{w}' \mathbf{x}_i))^2 \approx \sum_{i=1}^N (g'(\mathbf{w}'_{\text{old}} \mathbf{x}_i))^2 \left(\left(\mathbf{w}'_{\text{old}} \mathbf{x}_i + \frac{y_i - g(\mathbf{w}'_{\text{old}} \mathbf{x}_i)}{g'(\mathbf{w}'_{\text{old}} \mathbf{x}_i)} \right) - \mathbf{w}' \mathbf{x}_i \right)^2$$

Then \mathbf{w}'_{old} may be updated to \mathbf{w} using the closed form for weighted (by $(g'(\mathbf{w}'_{\text{old}} \mathbf{x}_i))^2$) no-intercept regression of

$$\left(\mathbf{w}'_{\text{old}} \mathbf{x}_i + \frac{y_i - g(\mathbf{w}'_{\text{old}} \mathbf{x}_i)}{g'(\mathbf{w}'_{\text{old}} \mathbf{x}_i)} \right)$$

on \mathbf{x}_i . (Presumably one must normalize the updated \mathbf{w} in order to preserve unit length property of the \mathbf{w} in order to maintain a stable scaling in the fitting.) The g and \mathbf{w} steps are iterated until convergence. Note that in the case where cubic smoothing spline smoothing is used in projection pursuit, g' will be evaluated as some explicit quadratic and in the case of locally weighted linear smoothing, form (95) will need to be differentiated in order to evaluate the derivative g' .

When $M > 1$, terms $g_m(\mathbf{w}'_m \mathbf{x})$ are added to a sum of such in a forward stage-wise fashion. HTF provide some discussion of details like readjusting previous g s (and perhaps \mathbf{w} s) upon adding $g_m(\mathbf{w}'_m \mathbf{x})$ to a fit, and the choice of M .

8 Highly Flexible Non-Linear Parametric Prediction Methods

8.1 Neural Network Regression

A multi-layer feed-forward neural network is a nonlinear map of $\mathbf{x} \in \mathfrak{R}^p$ to one or more outputs through the use of non-linear functions of linear combinations

of non-linear functions of linear combinations of ... non-linear functions of linear combinations of coordinates of \mathbf{x} . Figure 25 is a network diagram representation of a toy single hidden layer feed-forward neural net with 3 inputs, 2 hidden nodes, and 2 outputs.²⁴ The constants $x_0 = 1$ and $z_0 = 1$ allow for "biases" (i.e. constant terms) in the linear combinations (technically making them "affine" transformations rather than linear ones). The α and β parameters are sometimes called "weights."

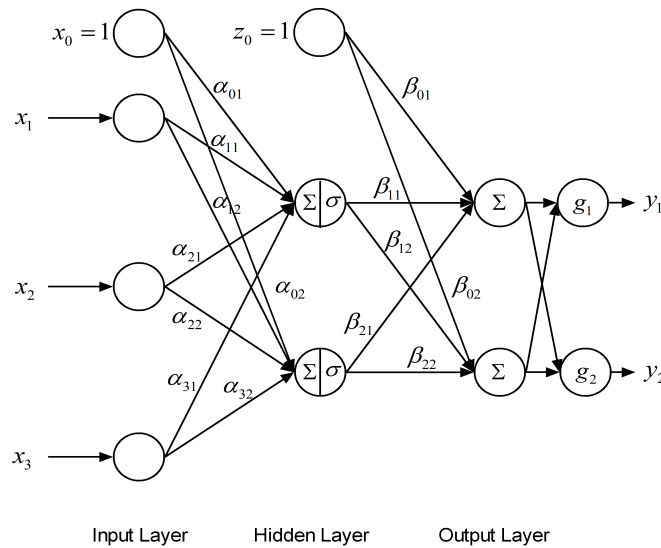


Figure 25: A Network Diagram Representation of a Single Hidden Layer Feed-forward Neural Net With 3 Inputs, 2 Hidden Nodes and 2 Outputs.

This diagram stands for a function of \mathbf{x} defined by setting

$$\begin{aligned} z_1 &= \sigma(\alpha_{01} \cdot 1 + \alpha_{11}x_1 + \alpha_{21}x_2 + \alpha_{31}x_3) \\ z_2 &= \sigma(\alpha_{02} \cdot 1 + \alpha_{12}x_1 + \alpha_{22}x_2 + \alpha_{32}x_3) \end{aligned}$$

and then

$$\begin{aligned} y_1 &= g_1(\beta_{01} \cdot 1 + \beta_{11}z_1 + \beta_{21}z_2, \beta_{02} \cdot 1 + \beta_{12}z_1 + \beta_{22}z_2) \\ y_2 &= g_2(\beta_{01} \cdot 1 + \beta_{11}z_1 + \beta_{21}z_2, \beta_{02} \cdot 1 + \beta_{12}z_1 + \beta_{22}z_2) \end{aligned}$$

In SEL/regression contexts, identity functions of a single one of the arguments are common and natural for the functions g .

Originally, the most common choice of functional form σ (the so-called "activation function") at hidden nodes was the (sigmoidal-shaped) logistic function²⁵

$$\sigma(u) = \frac{1}{1 + \exp(-u)}$$

²⁴Of course, much more complicated networks are possible, particularly ones with multiple hidden layers and many nodes on all layers.

²⁵Other functions with similar shapes, like the inverse standard normal cdf, were also used.

or the (completely equivalent in this context²⁶) hyperbolic tangent function

$$\sigma(u) = \tanh(u) = \frac{\exp(u) - \exp(-u)}{\exp(u) + \exp(-u)}$$

These functions are differentiable at $u = 0$, so that for small α s the functions of \mathbf{x} entering the g s in a single hidden layer network are nearly linear. For large α s the functions are nearly step functions. In light of the latter, it is not surprising that there are universal approximation theorems that guarantee that any continuous function on a compact subset of \mathfrak{R}^p can be approximated to any degree of fidelity with a single layer feed-forward neural net with enough nodes in the hidden layer. This is both a blessing and a curse. It promises that these forms are quite flexible. It also promises that there must be both overfitting and identifiability issues inherent in their use (the latter in addition to the identifiability issues already inherent in the symmetric nature of the functional forms assumed for the predictors).

More recently, sigmoidal forms for the activation function have declined in popularity. Instead, the hinge or positive part function

$$\sigma(u) = \max(u, 0) = u_+$$

is often used. In common parlance, this makes the hidden nodes "rectified linear units" (ReLU). Note that this choice makes functions of \mathbf{x} entering an output layer piece-wise linear and continuous (not at all an unreasonable form).

8.2 Neural Network Classification

In K -class classification problems, it is typical to use K output nodes and for $\mathbf{w} = (w_1, w_2, \dots, w_K)$ the vector of linear combinations of outputs from the final hidden layer, compute the outputs not simply using a single entry of \mathbf{w} for each, but rather using all entries. That is, it is typical to set K outputs to be

$$g_k(\mathbf{w}) = \frac{\exp(w_k)}{\sum_{l=1}^K \exp(w_l)} \quad (100)$$

This vector function of (vector) \mathbf{w} is usually referred to as the "softmax" function, and produces a probability vector as output. Its entries serve as estimates of class probabilities for the given vector of inputs. The 0-1 loss classifier corresponding to this set of estimated class probabilities is then

$$\hat{f}(\mathbf{x}) = \arg \max_k g_k$$

(where it is understood that the k th probability, g_k , depends upon the input \mathbf{x} through the neural net compositions of functions and the final use of the softmax function).

²⁶This is because $\tanh(u) = 2 \left(\frac{1}{1 + \exp(-2u)} \right) - 1$.

8.3 Fitting Neural Networks

8.3.1 The Back-Propagation Algorithm

The most common fitting algorithm for neural networks is something called the "back-propagation algorithm" or the "delta rule." It is simply a gradient descent algorithm for the entire set of weights involved in making the outputs (in the simple case illustrated in Figure 25, the α s and β s). Rather than labor through the nasty notational issues required to completely detail such an algorithm, we will here only lay out the heart of what is needed.

For a training set of size N , loss $L(\hat{\mathbf{f}}(\mathbf{x}_i), y_i)$ incurred for input case i when the K predictions $\hat{f}_k(\mathbf{x}_i)$ are made (corresponding to the K output nodes) and a sum of such losses is to be minimized, if one can find the partial derivatives of the coordinates of $\hat{\mathbf{f}}(\mathbf{x})$ with respect to the weights, the chain rule will give the partials of the total loss and allow iterative search in the direction of a negative gradient of the total loss. So we begin with description of how to find partials for $\hat{f}_k(\mathbf{x})$, a coordinate of the fitted output vector.

Consider a neural network with H layers of hidden nodes indexed by $h = 1, 2, \dots, H$ beginning with the layer immediately before the output layer and proceeding (right to left in a diagram like Figure 25) to the one that is built from linear combinations of the coordinates of \mathbf{x} . We'll use the notation m_h for the number of nodes in layer h , *including* a node representing the "bias" input 1 (represented by $x_0 = 1$ and $z_0 = 1$ in Figure 25). For a real-valued activation function of a single real variable σ , define a vector-valued function $\sigma_m : \mathfrak{R}^m \rightarrow \mathfrak{R}^m$ by

$$\sigma_m(u_1, u_2, \dots, u_m) = (\sigma(u_1), \sigma(u_2), \dots, \sigma(u_m))$$

In what follows (for purposes of reducing notational clutter) we will abuse notation somewhat and not subscript σ_m , but rather write only σ , leaving it to the reader to recall that σ outputs vectors of the same dimension as its argument. And it will be convenient to presume that both the input and output of a σ are *row* vectors.

Then for \mathbf{A}^H a $(p+1) \times (m_H - 1)$ matrix of (weight) parameters we can represent the relationship between the input \mathbf{x} and vector of values (say \mathbf{z}_H) in the last hidden layer by

$$\mathbf{z}'_H = \left(1, \sigma\left((1, \mathbf{x}') \mathbf{A}^H\right)\right)$$

Next, for \mathbf{A}^{H-1} an $m_H \times (m_{H-1} - 1)$ matrix of parameters we may represent the relationship between the vectors of values in the last and next to last hidden layers by

$$\mathbf{z}'_{H-1} = \left(1, \sigma\left(\mathbf{z}'_H \mathbf{A}^{H-1}\right)\right)$$

and so on to the $h = 1$ case of (\mathbf{A}^h an $m_{h+1} \times (m_h - 1)$ matrix of parameters)

$$\mathbf{z}'_h = \left(1, \sigma\left(\mathbf{z}'_{h+1} \mathbf{A}^h\right)\right) \tag{101}$$

Then for \mathbf{A}^0 an $m_1 \times K$ matrix of parameters and g_k a function of K real variables, the k th coordinate of the output is

$$g_k(\mathbf{z}'_1 \mathbf{A}^0) \quad (102)$$

This series of relationships allows (via what is known as a "forward pass" through them) the computation of \mathbf{z} s and predictions for a fixed set of coefficients collected in the \mathbf{A} s and an input vector \mathbf{x} . Then partial derivatives of the k th coordinate of the response (at that input and set of coefficients) can be found via the "backward pass" based on the K partials of g_k , the derivative of σ , the recursions above, and the results of the forward pass.

For example, for $g_k^{(l)}$ the partial of the function g_k with respect to its l th entry, the partial derivative of the k th coordinate of the prediction with respect to the (i, j) entry of \mathbf{A}^0 is from relationship (102) and the chain rule

$$g_k^{(j)}(\mathbf{z}'_1 \mathbf{A}^0) \cdot z_{1i}$$

Further, since using form (102) and the $h = 1$ version of form (101) the k th coordinate of the prediction is

$$g_k((1, \sigma(\mathbf{z}'_2 \mathbf{A}^1)) \mathbf{A}^0)$$

writing a_{ij}^1 for the (i, j) entry of \mathbf{A}^1 , the chain rule implies that (with \mathbf{A}_l^0 the l th column of \mathbf{A}^0) the partial derivative of the k th coordinate of the prediction with respect to a_{ij}^1 is

$$\begin{aligned} & \sum_{l=1}^K g_k^{(l)}((1, \sigma(\mathbf{z}'_2 \mathbf{A}^1)) \mathbf{A}^0) \frac{\partial}{\partial a_{ij}^1} ((1, \sigma(\mathbf{z}'_2 \mathbf{A}^1)) \mathbf{A}^0)_l \\ &= \sum_{l=1}^K g_k^{(l)}((1, \sigma(\mathbf{z}'_2 \mathbf{A}^1)) \mathbf{A}^0) \frac{\partial}{\partial a_{ij}^1} ((1, \sigma(\mathbf{z}'_2 \mathbf{A}^1)) \mathbf{A}_l^0) \\ &= \sum_{l=1}^K g_k^{(l)}((1, \sigma(\mathbf{z}'_2 \mathbf{A}^1)) \mathbf{A}^0) \sum_{k=1}^K a_{kl}^0 \frac{\partial}{\partial a_{ij}^1} (1, \sigma(\mathbf{z}'_2 \mathbf{A}^1))_k \\ &= \sum_{l=1}^K g_k^{(l)}((1, \sigma(\mathbf{z}'_2 \mathbf{A}^1)) \mathbf{A}^0) a_{jl}^0 \frac{\partial}{\partial a_{ij}^1} \sigma(\mathbf{z}'_2 \mathbf{A}^1) \\ &= \sum_{l=1}^K g_k^{(l)}((1, \sigma(\mathbf{z}'_2 \mathbf{A}^1)) \mathbf{A}^0) \sigma'(\mathbf{z}'_2 \mathbf{A}^1) a_{jl}^0 z_{2i} \end{aligned}$$

"and so on" for other \hat{y}_k s and a_{ij}^h s.

In general one is faced with the functional form for the k th coordinate of the output

$$g_k \left(\left(1, \sigma \left(\left(1, \sigma \left(\left(1, \sigma \left(\left(1, \sigma \left((1, \mathbf{x}') \mathbf{A}^H \right) \mathbf{A}^{H-1} \right) \right) \right) \right) \right) \right) \right) \mathbf{A}^2 \right) \mathbf{A}^1 \right) \mathbf{A}^0$$

made by successive compositions using the activation function and linear combinations with coefficients in the matrices \mathbf{A}^h , from which partials $\frac{\partial \hat{y}_k}{\partial a_{ij}^h}$ are obtainable in the style above, by repeatedly using the chain rule. No doubt some appropriate use of vector calculus and corresponding notation could improve the looks of these expressions and recursions can be developed, but what is needed should be clear. Further, in many contexts numerical approximation of these partials may be the most direct and efficient means of obtaining them.

Then for loss $L(\hat{\mathbf{f}}, y)$ let

$$L_k(\hat{\mathbf{f}}, y) = \frac{\partial}{\partial \hat{f}_k} L(\hat{\mathbf{f}}, y)$$

For a an element of one of the \mathbf{A}^h matrices, the partial derivative of the contribution of case i to a total loss with respect to it is

$$\sum_{k=1}^K L_k(\hat{\mathbf{f}}(\mathbf{x}_i), y_i) \frac{\partial}{\partial a} g_k(\mathbf{z}'_1(\mathbf{x}_i) \mathbf{A}^0)$$

(for $\mathbf{z}'_1(\mathbf{x}_i)$ the set of values from the final hidden nodes and partials found as above) and the partial derivative of the total loss with respect to it is

$$D(a) = \sum_{i=1}^N \sum_{k=1}^K L_k(\hat{\mathbf{f}}(\mathbf{x}_i), y_i) \frac{\partial}{\partial a} g_k(\mathbf{z}'_1(\mathbf{x}_i) \mathbf{A}^0)$$

The gradient of the total loss as a function of the matrices of weights then has entries $D(a)$ and an iterative search to optimize total loss with a current set of iterates a_{current} can produce new iterates

$$a_{\text{new}} = a_{\text{current}} - \gamma D(a_{\text{current}}) \quad (103)$$

for some "learning rate" $\gamma > 0$.

Of course, in SEL/univariate regression contexts, it is common to have $K = 1$ and take $L(\hat{f}, y) = (\hat{f} - y)^2$. In K -class classification models, it seems most common to use a K -dimensional output $\hat{\mathbf{g}} = (g_1(\mathbf{z}'_1 \mathbf{A}^0), g_2(\mathbf{z}'_1 \mathbf{A}^0), \dots, g_K(\mathbf{z}'_1 \mathbf{A}^0))$ with the "softmax" g_k as defined in display (100) and to employ the cross-entropy loss

$$L(\hat{\mathbf{g}}, y) = - \sum_{k=1}^K I[y = k] \ln g_k(\mathbf{x})$$

There are various possibilities for regularization of the ill-posed fitting problem for neural nets, ranging from the fairly formal and rational to the very informal and ad hoc. One possibility is to employ "stochastic gradient descent" and newly choose a random subset of the training set for use at each iteration of fitting. (It is popular to even go so far in this regard as to employ only a single case at each iteration.) Another common approach is to simply use an iterative fitting algorithm and "stop it before it converges." We proceed to briefly discuss more formal regularization.

8.3.2 Formal Regularization of Fitting

Suppose that the various coordinates of the input vectors in the training set have been standardized and one wants to regularize the fitting of a neural net. One possible way of proceeding is to define a penalty function like

$$J(\mathbf{A}) = \sum_{h=0}^H \sum_{i,j} (a_{ij}^h)^2 \quad (104)$$

for \mathbf{A} standing for the entire set of weights in $\mathbf{A}^0, \mathbf{A}^1, \dots, \mathbf{A}^H$ (it is not absolutely clear whether one really wants to include the weights on the "bias" terms in the neural net sums in (104)) and seek not to partially optimize the total training set loss $\sum_{i=1}^N L(\hat{\mathbf{f}}_{\mathbf{A}}(\mathbf{x}_i), y_i)$ but rather to fully optimize

$$\sum_{i=1}^N L(\hat{\mathbf{f}}_{\mathbf{A}}(\mathbf{x}_i), y_i) + \lambda J(\mathbf{A}) \quad (105)$$

for a $\lambda > 0$. By modifying the recursion (103) to

$$a_{\text{new}} = a_{\text{current}} - \gamma(D(a_{\text{current}}) + 2\lambda a_{\text{current}})$$

one arrives at an appropriate gradient descent algorithm for optimizing the penalized training loss (105). (Potentially, an appropriate value for λ might be chosen based on cross-validation.)

Something that may at first seem quite different would be to take a Bayesian point of view. For example, with a univariate regression model for outputs

$$y_i = f(\mathbf{x}_i|\mathbf{A}) + \epsilon_i$$

for the ϵ_i iid $N(0, \sigma^2)$, a likelihood is simply

$$l(\mathbf{A}, \sigma^2) = \prod_{i=1}^N h(y_i|f(\mathbf{x}_i|\mathbf{A}), \sigma^2)$$

for $h(\cdot|\mu, \sigma^2)$ the normal pdf. If then $g(\mathbf{A}, \sigma^2)$ specifies a prior distribution for \mathbf{A} and σ^2 , a posterior for (\mathbf{A}, σ^2) has density proportional to

$$l(\mathbf{A}, \sigma^2) g(\mathbf{A}, \sigma^2)$$

For example, one might well assume that *a priori* the a s are iid $N(0, \eta^2)$ (where small η^2 will provide regularization and it is again unclear whether one wants to include the a s corresponding to bias terms in such an assumption or to instead provide more diffuse priors for them, like improper "Uniform $(-\infty, \infty)$ " or at least large variance normal ones). A standard improper prior for σ^2 is $\ln \sigma \sim \text{Uniform}(-\infty, \infty)$. In any case, whether improper or proper, abuse notation and write $g(\sigma^2)$ for a prior density for σ^2 .

Then with independent mean 0 variance η^2 priors for all the weights (except possibly the ones for bias terms that might be given Uniform $(-\infty, \infty)$ priors) one has

$$\begin{aligned} & \ln (l(\mathbf{A}, \sigma^2) g(\mathbf{A}, \sigma^2)) \\ & \propto -NK \ln (\sigma) - \frac{1}{2\sigma^2} \sum_{i=1}^N (y_i - f(\mathbf{x}_i|\mathbf{A}))^2 - \frac{1}{2\eta^2} J(\mathbf{A}) + \ln g(\sigma^2) \\ & = -NK \ln (\sigma) + \ln g(\sigma^2) - \frac{1}{\sigma^2} \left(\sum_{i=1}^N (y_i - f(\mathbf{x}_i|\mathbf{A}))^2 + \frac{\sigma^2}{\eta^2} J(\mathbf{A}) \right) \quad (106) \end{aligned}$$

(flat improper priors for the bias weights correspond to the absence of terms for them in the sums for $J(\mathbf{A})$ in form (104)). This recalls display (105) and suggests that appropriate λ for regularization can be thought of as a variance ratio of "observation variance" and prior variance for the weights.

It's fairly clear how to define Metropolis-Hastings-within-Gibbs algorithms for sampling from $l(\mathbf{A}, \sigma^2) g(\mathbf{A}, \sigma^2)$. But it seems that typically the high dimensionality of the parameter space combined with the symmetry-derived multimodality of the posterior will prevent one from running an MCMC algorithm long enough to fully detail the posterior. It also seems unlikely however, that detailing the posterior is really necessary or even desirable. Rather, one might simply run the MCMC algorithm, monitoring the values of $l(\mathbf{A}, \sigma^2) g(\mathbf{A}, \sigma^2)$ corresponding to the successively randomly generated MCMC iterates. An MCMC algorithm will spend much of its time where the corresponding posterior density is large and we can expect that a long MCMC run will identify a nearly modal value for the posterior. Rather than averaging neural nets according to the posterior, one might instead use as a predictor a neural net corresponding to a parameter vector (at least locally) maximizing the posterior.

Notice that one might even take the parameter vector in an MCMC run with the largest $l(\mathbf{A}, \sigma^2) g(\mathbf{A}, \sigma^2)$ value and for a grid of σ^2 values around the empirical maximizer use the back-propagation algorithm modified to fully optimize

$$\sum_{i=1}^N (y_i - f(\mathbf{x}_i|\mathbf{A}))^2 + \frac{\sigma^2}{\eta^2} J(\mathbf{A})$$

over choices of \mathbf{A} . This, in turn, could be used with relationship (106) to perhaps improve somewhat the result of the MCMC "search."

8.4 Convolutional Neural Networks

An application of neural network type ideas that has received much recent attention is that of image classification. We will here provide a short introduction to the area. Not surprisingly, success in this realm seems to rely as much upon ideas from image processing as upon ideas from prediction.

Mathematically, a grey-scale image is typically represented by an $L \times M$ matrix $\mathbf{X} = [x_{lm}]$ where each $x_{lm} \in \{0, 1, 2, \dots, 254, 255\}$ represents a brightness at location (l, m) . A color image is often represented by 3 matrices $\mathbf{X}^r = [x_{lm}]$, $\mathbf{X}^g = [x_{lm}]$, and $\mathbf{X}^b = [x_{lm}]$ (again all with integer entries in $\{0, 1, 2, \dots, 254, 255\}$) representing intensities in red, green, and blue "channels." The standard machine learning problem is to (based on a training set of N images \mathbf{X}_i or $[\mathbf{X}^r, \mathbf{X}^g, \mathbf{X}^b]_i$ with corresponding class identities $y_i \in \{1, 2, \dots, K\}$) produce a classifier. (For example, a standard test problem is "automatic" recognition of hand-written digits 0 through 9.)

Simple convolutional neural networks with H hidden layers and a softmax output layer producing class probabilities are successive compositions of more or less natural linear and non-linear operations that might be represented as follows. For σ^H operating on \mathbf{X} using some set of real number parameters \mathbf{A}^H to produce some multivariate output (we will describe below some kinds of things that are popularly used) a "deepest layer" of the convolutional neural net produces

$$\mathbf{Z}^H \equiv \sigma^H(\mathbf{X}, \mathbf{A}^H) \quad (107)$$

Then applying another set of operations σ^{H-1} to the result (107) using some set of parameters \mathbf{A}^{H-1} , the next layer of values in the convolutional neural net is produced as

$$\mathbf{Z}^{H-1} = \sigma^{H-1}(\mathbf{Z}^H, \mathbf{A}^{H-1})$$

and so on, with

$$\mathbf{Z}^h = \sigma^h(\mathbf{Z}^{h+1}, \mathbf{A}^h) \quad (108)$$

for $h = H, H-1, \dots, 1$ where σ^1 is \Re^K -valued (the top layer of hidden values is a K -vector). Then, with g the softmax function the output K -vector of class probabilities is

$$g(\mathbf{Z}^1)$$

In multi-channel cases, it seems common to develop separate series of compositions based on \mathbf{X}^r , \mathbf{X}^g , and \mathbf{X}^b and to bring them together only in the top or top few levels of this kind of hierarchy.

Variants of this basic structure are possible and have been used. For example, it is sometimes done to make a "direct connection" between layer h and one deeper than layer $h+1$. That is the option to employ a form

$$\mathbf{Z}^h = \sigma^h(\mathbf{Z}^{h+1}, \mathbf{Z}^{h+j}, \mathbf{A}^h)$$

for some $j > 1$ or even a form

$$\mathbf{Z}^h = \sigma^h(\mathbf{Z}^{h+1}, \mathbf{X}, \mathbf{A}^h)$$

making a direct connection to the input layer is sometimes employed. (Obviously, even more complicated schemes are possible.)

Most of what we have said thus far in this section is not really special to the problem of image classification (and could serve as a high-level introduction to general neural net predictors). What sets the "convolutional" neural network field apart from "generic" neural network practice is the image-processing-inspired forms employed in the functions σ^h . The most fundamental form is one that applies "linear filters" to images followed by some nonlinear operation. This creates what is commonly called a "convolution" layer.

To make the idea of a convolutional layer precise, consider the following. Let \mathbf{F} be an $R \times C$ matrix. Typically this matrix is much smaller than the image and square (at least when "horizontal" and "vertical" resolutions in the images are the same), and R and C are often odd. One can then make from \mathbf{F} and \mathbf{X} a new matrix $\mathbf{F} \otimes \mathbf{X}$ of dimension $(L - R + 1) \times (M - C + 1)$ with entries

$$(\mathbf{F} \otimes \mathbf{X})_{ij} = \sum_{a=1}^R \sum_{b=1}^C f_{ab} x_{(i+a-1), (j+b-1)} \quad (109)$$

A natural way to think about this operation is to align an $R \times C$ integer grid with values in \mathbf{F} on the grid points with a (larger) corresponding grid for the image \mathbf{X} , setting the upper left $(1, 1)$ corner of the \mathbf{F} grid at the (i, j) location on the \mathbf{X} grid, and to then sum products of aligned matrix entries. The entries of \mathbf{F} serve as weights on the values in the $R \times C$ part of the image aligned with the filter matrix. Figure 26 illustrates this process for a simple case where \mathbf{F} is 3×3 (and so ultimately $\mathbf{F} \otimes \mathbf{X}$ has 2 fewer columns and 2 fewer rows than \mathbf{X} and is thus $(L - 2) \times (M - 2)$).

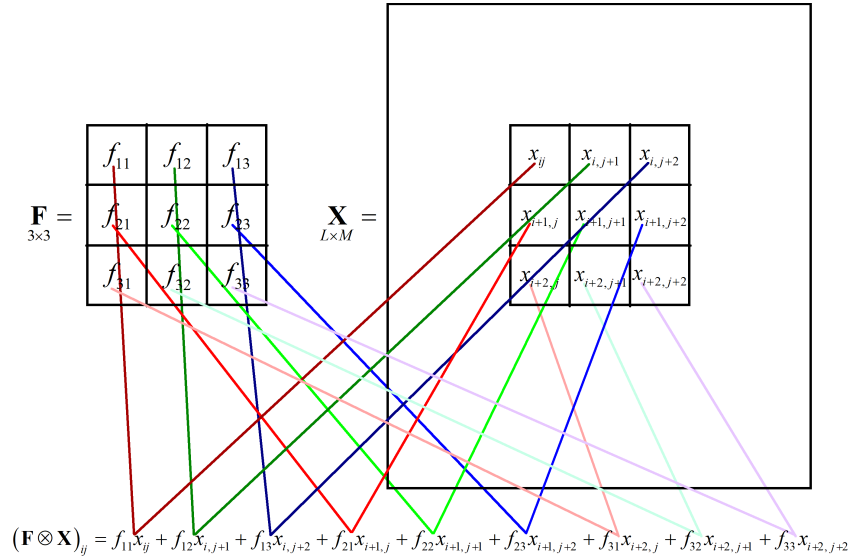


Figure 26: Illustration of the use of the 3×3 filter matrix \mathbf{F} with $L \times M$ image matrix \mathbf{X} to produce the $(L - 2) \times (M - 2)$ matrix $\mathbf{F} \otimes \mathbf{X}$.

This convolution operation is linear and it is typical practice to introduce non-linearity by following convolution operations in a layer with the hinge function $\max(u, 0)$ applied to each element u of the resulting matrix. Sometimes people (apparently wishing to not lose rows and columns in the convolution process) "0 pad" an image with extra rows and columns of 0s before doing the convolution—a practice that strikes this author as lacking sound rationale.

Multiple convolutions are typically created in a single convolution layer. Sometimes the filter matrices are filled with parameters to be determined in fitting (i.e. are part of \mathbf{A}^h in the representation (108)). But they can also be fixed matrices created for specific purposes. For example the 3×3 matrices

$$\mathbf{S}^{\text{vert}} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{S}^{\text{horiz}} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

are respectively the vertical and horizontal Sobel filter matrices, commonly used in image processing when searching for edges of objects or regions. And various "blurring" filters (ordinary arithmetic averaging across a square of pixels and weighted averaging done according to values of a Gaussian density set at the center of an integer grid) are common devices meant to suppress noise in an image.

As multiple layers each with multiple new convolutions are created, there is potential explosion of the total dimensionality of the sets of \mathbf{Z}^h and \mathbf{A}^h . Two devices for controlling that explosion are the notions of sampling and pooling to reduce the size of a \mathbf{Z} . First, instead of creating and subsequently using an entire filtered image $\mathbf{F} \otimes \mathbf{X}$, one can use only every s th row and column. In such a "sampling" operation s is colloquially known as the "stride." Roughly speaking, this reduces the size of a \mathbf{Z} by a factor of s^2 . Another possibility is to choose some block size, of size say $s \times t$, and divide an $L \times M$ image into roughly

$$\left(\frac{L}{s}\right) \left(\frac{M}{t}\right)$$

non-overlapping blocks, within a block applying a "pooling" rule like "simple averaging" or "maximum value." One then uses the rectangular array of these pooled values as a layer output. This, of course, reduces the size of a \mathbf{Z} by a factor of roughly st . It seems common to apply one of these ideas after each one or few convolution layers in a network, and especially before reaching the top and final one or few layers. The final hidden layers of a convolutional neural net are of the "ordinary" type described earlier and if the dimensionality of their inputs are too large, numerical and fitting problems will typically ensue.

8.5 Recurrent Neural Networks

Another context where neural network ideas have found application is that of (non-linear) time series prediction. That is, vectors of inputs and outputs are sometimes indexed with time order and one expects information from previous

periods to be of help in predicting response at the current one. To give some sense of what can be done, consider a generalization of the toy single hidden layer feed-forward neural net with 3 inputs, 2 hidden nodes, and 2 outputs used in Section 8.1. Where input/output pairs (\mathbf{x}, \mathbf{y}) with $\mathbf{x} \in \mathfrak{R}^3$ and $\mathbf{y} \in \mathfrak{R}^2$ are indexed by (time) integer t , the notion of recurrent neural network practice is to allow values z_{1t} and z_{2t} at the hidden nodes to depend not only upon \mathbf{x}_t but also upon z_{1t-1} and z_{2t-1} and/or \mathbf{y}_{t-1} .

A so-called Elman Network replaces the basic expressions for moving from input to hidden layer in Section 8.1 with

$$\begin{aligned} z_{1t} &= \sigma(\alpha_{01} \cdot 1 + \alpha_{11}x_{1t} + \alpha_{21}x_{2t} + \alpha_{31}x_{3t} + \alpha_{11}^*z_{1t-1} + \alpha_{21}^*z_{2t-1}) \\ z_{2t} &= \sigma(\alpha_{02} \cdot 1 + \alpha_{12}x_{1t} + \alpha_{22}x_{2t} + \alpha_{32}x_{3t} + \alpha_{12}^*z_{1t-1} + \alpha_{22}^*z_{2t-1}) \end{aligned}$$

and a Jordan Network replaces them with

$$\begin{aligned} z_{1t} &= \sigma(\alpha_{01} \cdot 1 + \alpha_{11}x_{1t} + \alpha_{21}x_{2t} + \alpha_{31}x_{3t} + \alpha_{11}^*y_{1t-1} + \alpha_{21}^*y_{2t-1}) \\ z_{2t} &= \sigma(\alpha_{02} \cdot 1 + \alpha_{12}x_{1t} + \alpha_{22}x_{2t} + \alpha_{32}x_{3t} + \alpha_{12}^*y_{1t-1} + \alpha_{22}^*y_{2t-1}) \end{aligned}$$

These are obviously some kind of non-linear auto-regressive relationships and introduce additional weights α^* that must be fit in order to apply the prediction methodology.

It's obvious that once one opens this line of thinking many more complicated forms are possible. Forms for values at current hidden nodes could be postulated to depend explicitly on values at hidden nodes or outputs further in the past than period $t-1$. Both values at hidden nodes and outputs could be involved. Etc. Fitting algorithms based on gradient descent are tailored to the particular recurrence relationships employed.

8.6 Radial Basis Function Networks

Section 6.7 of HTF considers the use of the kind of kernels applied in kernel smoothing as basis functions. That is, for

$$\mathcal{K}_\lambda(\mathbf{x}, \boldsymbol{\xi}) = D\left(\frac{\|\mathbf{x} - \boldsymbol{\xi}\|}{\lambda}\right)$$

one might consider fitting nonlinear predictors of the form

$$f(\mathbf{x}) = \beta_0 + \sum_{j=1}^M \beta_j \mathcal{K}_\lambda(\mathbf{x}, \boldsymbol{\xi}_j) \quad (110)$$

where each basis element has prototype parameter $\boldsymbol{\xi}$ and scale parameter λ . A common choice of D for this purpose is the standard normal pdf.

A version of this with fewer parameters is obtained by restricting to cases where $\lambda_1 = \lambda_2 = \dots = \lambda_M = \lambda$. This restriction, however, has the potentially unattractive effect of forcing "holes" or regions of \mathfrak{R}^p where (in each) $f(\mathbf{x}) \approx \beta_0$, including all "large" \mathbf{x} . A way to replace this behavior with potentially differing

values in the former "holes" and directions of "large" \mathbf{x} is to replace the basis functions

$$\mathcal{K}_\lambda(\mathbf{x}, \boldsymbol{\xi}_j) = D\left(\frac{\|\mathbf{x} - \boldsymbol{\xi}_j\|}{\lambda}\right)$$

with normalized versions

$$h_j^\lambda(\mathbf{x}) = \frac{D(\|\mathbf{x} - \boldsymbol{\xi}_j\|/\lambda)}{\sum_{k=1}^M D(\|\mathbf{x} - \boldsymbol{\xi}_k\|/\lambda)}$$

to produce a form

$$f(\mathbf{x}) = \beta_0 + \sum_{j=1}^M \beta_j h_j^\lambda(\mathbf{x}) \quad (111)$$

The fitting of form (110) by choice of $\beta_0, \beta_1, \dots, \beta_M, \boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \dots, \boldsymbol{\xi}_M, \lambda_1, \lambda_2, \dots, \lambda_M$ or form (111) by choice of $\beta_0, \beta_1, \dots, \beta_M, \boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \dots, \boldsymbol{\xi}_M, \lambda$ is fraught with all the problems of over-parameterization and lack of identifiability associated with neural networks.

Another way to use radial basis functions to produce flexible functional forms is to replace the forms $\sigma(\alpha_{0m} + \boldsymbol{\alpha}'_m \mathbf{x})$ in a neural network with forms $\mathcal{K}_{\lambda_m}(\boldsymbol{\alpha}'_m \mathbf{x}, \boldsymbol{\xi}_m)$ or $h_m^\lambda(\boldsymbol{\alpha}'_m \mathbf{x})$.

9 Prediction Methods Based on Rectangles: Trees and PRIM

This section begins something genuinely new to our discussion. That is the search for good predictors that are constant on p -dimensional "rectangles" in the input space, that is on subsets of \mathfrak{R}^p of the form

$$R = \{\mathbf{x} \in \mathfrak{R}^p | a_1 < x_1 < b_1 \text{ and } a_2 < x_2 < b_2 \dots \text{ and } a_p < x_p < b_p\}$$

for (possibly infinite) values $a_j < b_j$ for $j = 1, 2, \dots, p$. The basic idea is that if the values a_j and b_j can be chosen so that y s corresponding to vectors of inputs \mathbf{x} in a training set in a particular rectangle are "homogeneous," then a corresponding SEL predictor using training set "rectangle mean responses" or a 0-1 loss classifier using training set "rectangle majority classes" might be approximately optimal.²⁷

The search for good predictors constant on rectangles is fundamentally an algorithmic matter, rather than something that will have a nice closed form representation (it is not like ridge regression for example). But (provided "fast" and "effective" algorithms can be identified) it has things that make it very attractive. For one thing, there is complete **invariance to monotone transformation of numerical features**. It is irrelevant to searches for good

²⁷This is essentially the same motivation provided for nearest neighbor rules in Section 1.3.3.

boundaries for rectangles whether a coordinate of the input \mathbf{x} is expressed on an "original" scale or a log scale or on another (monotone transform of the original scale). The same predictor/predictions will result. This is a very attractive and powerful feature and is no doubt partly responsible for the popularity of rectangle-based predictors as building blocks for more complicated methods (like "boosting trees").

The structure of predictors constant on rectangles is also an intuitively appealing one, easily explained and understood. This helps make them very popular with non-technical consumers of predictive analytics.

In this section we consider two rectangle-based prediction methods, the first (CART) using binary tree structures and the second (PRIM) employing a kind of "bump-hunting" logic.

9.1 Regression and Classification Trees (CART)

The common acronym for this methodology is CART (classification and regression trees) and classification trees are sometimes referred to as "decision trees." Here we'll first consider the SEL/regression version and then the classification version.

9.1.1 Regression Trees

We consider a forward-selection/"greedy" algorithm for inventing predictions constant on p -dimensional rectangles, by successively looking for an optimal binary split of a single one of an existing set of rectangles. Define

$$a_j = \min_{i=1,2,\dots,N} x_{ij} \text{ and } b_j = \max_{i=1,2,\dots,N} x_{ij}$$

Begin with the rectangle in \mathfrak{R}^p

$$R = \prod_{j=1}^p [a_j, b_j] = \{\mathbf{x} \in \mathfrak{R}^p \mid \text{each } a_j \leq x_j \leq b_j\}$$

and look for an index j_1 and a value $a_{j_1} < s_1 < b_{j_1}$ (with $s_1 \neq x_{ij_1}$ for any i) so that splitting the initial rectangle at $x_{j_1} = s_1$ (to produce the two sub-rectangles $R \cap \{\mathbf{x} \in \mathfrak{R}^p \mid x_{j_1} \leq s_1\}$ and $R \cap \{\mathbf{x} \in \mathfrak{R}^p \mid x_{j_1} > s_1\}$) so that the resulting two rectangles minimize

$$SSE = \sum_{\text{rectangles } i \text{ with } \mathbf{x}_i \text{ in the rectangle}} \sum (y_i - \bar{y}_{\text{rectangle}})^2$$

One then splits (optimally) one of the (now) two rectangles on some variable x_{j_2} at some s_2 (with $s_2 \neq x_{ij_2}$ for any i) etc.

Where l rectangles in \mathfrak{R}^p (say R_1, R_2, \dots, R_l) have been created, and

$$m(\mathbf{x}) = \text{the index of the rectangle to which } \mathbf{x} \text{ belongs}$$

the corresponding SEL tree predictor is

$$\hat{f}_l(\mathbf{x}) = \frac{1}{\# \text{ training input vectors } \mathbf{x}_i \text{ in } R_m(\mathbf{x})} \sum_{\substack{i \text{ with } \mathbf{x}_i \\ \text{in } R_m(\mathbf{x})}} y_i$$

and in this notation the training error is

$$SSE = \sum_{i=1}^N (y_i - \hat{f}_l(\mathbf{x}_i))^2$$

or the corresponding mean squared prediction error

$$\overline{\text{err}} = \frac{1}{N} SSE$$

If one is to continue beyond l rectangles, one then looks for a value s_l to split one of the existing rectangles R_1, R_2, \dots, R_l on some x_{j_l} and thereby produce the greatest reduction in SSE . (We note that there is no guarantee that after l splits one will have the best (in terms of SSE) possible set of $l+1$ rectangles.)

Any series of binary splits of rectangles can be represented graphically as a binary tree, each split represented by a node where there is a fork and each final rectangle by an end node. It is convenient to discuss rectangle-splitting (and "unsplitting") in terms of operations on corresponding binary trees, and henceforth we adopt this language. Figures 27 and 28 provide three representations of the same hypothetical regression tree with $p = 2 \dots$ a predictor constant on rectangles in \mathbb{R}^2 .

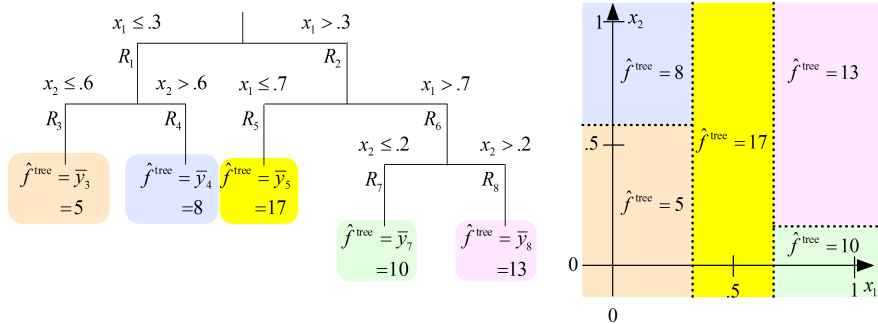


Figure 27: Two representations of a hypothetical tree predictor for $p = 2$.

The basic formulation of the tree-growing method here employs "one-step-at-a-time"/"greedy" (unable to defer immediate reward for the possibility of later success) methods. Like all such methods, they are not guaranteed to follow paths through the set of trees that ever get to "best" ones since they are "myopic," never considering what might be later in a search, if a current step were taken that provides little immediate payoff.

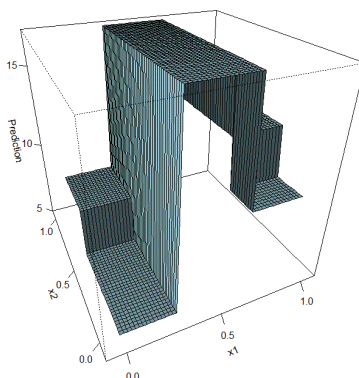


Figure 28: A third representation of the hypothetical $p = 2$ tree predictor portrayed in Figure 27.

A regression tree example (essentially suggested by Mark Culp) dramatically illustrates this limitation. Consider a $p = 3$ case where $x_1 \in \{0, 1\}$, $x_2 \in \{0, 1\}$, and $x_3 \in [0, 1]$. In fact, suppose that x_1 and x_2 are iid Bernoulli(.5) independent of x_3 that is Uniform(0, 1). Then suppose that conditional on (x_1, x_2, x_3) the output y is $N(\mu(x_1, x_2, x_3), 1)$ for

$$\mu(x_1, x_2, x_3) = 1000 \cdot I[x_1 = x_2] + x_3$$

For a big training sample iid from this joint distribution, all branching will typically be done on the continuous variable x_3 , completely missing the fundamental fact that it is the (joint) behavior of (x_1, x_2) that drives the size of y . (This example also supports the conventional wisdom that as presented the splitting algorithm "favors" splitting on continuous variables over splitting on values of discrete ones.)

9.1.2 Classification Trees

The "classification trees" version of this material is very similar to the continuous y (SEL) regression tree version. One needs only to define an empirical loss to associate with a given tree parallel to SSE used above. To that end, note that in a K -class problem (where y takes values in $\mathcal{G} = \{1, 2, \dots, K\}$) corresponding to a particular rectangle R_m is the fraction of training vectors with classification k ,

$$\widehat{p}_{mk} = \frac{1}{\# \text{ training input vectors in } R_m} \sum_{\substack{i \text{ with } \mathbf{x}_i \\ \text{in } R_m}} I[y_i = k]$$

and a plausible \mathcal{G} -valued predictor based on l rectangles is

$$\hat{f}_l(\mathbf{x}) = \arg \max_{k \in \mathcal{G}} \widehat{p}_{m(\mathbf{x})k}$$

the class that is most heavily represented in the rectangle to which \mathbf{x} belongs.²⁸ The empirical misclassification rate for this predictor (that can be used as a rectangle-splitting criterion) is

$$\overline{\text{err}} = \frac{1}{N} \sum_{i=1}^N I [y_i \neq \hat{f}_l(\mathbf{x}_i)] = \frac{1}{N} \sum_{m=1}^l N_m (1 - \widehat{p}_{mk(m)})$$

where $N_m = \#$ training input vectors in R_m , and $k(m) = \arg \max_{k \in \mathcal{G}} \widehat{p}_{mk}$. Two

other popular splitting criteria are "the Gini index"

$$\overline{\text{err}} = \frac{1}{N} \sum_{m=1}^l N_m \left(\sum_{k=1}^K \widehat{p}_{mk} (1 - \widehat{p}_{mk}) \right)$$

and the so-called "cross entropy"

$$\overline{\text{err}} = -\frac{1}{N} \sum_{m=1}^l N_m \left(\sum_{k=1}^K \widehat{p}_{mk} \ln(\widehat{p}_{mk}) \right)$$

These latter two criteria are average (across rectangles) measures of "purity" (near degeneracy) of training set response distributions in the rectangles. Upon adopting one of these forms to replace SSE in the regression tree discussion, one has a classification tree methodology. HTF suggest using the Gini index or cross entropy for tree growing and any of the indices (but most typically the empirical misclassification rate) for tree pruning according to cost-complexity (to be discussed next).

9.1.3 Optimal Subtrees

It is, of course, possible to continue splitting rectangles/adding branches to a tree until every distinct \mathbf{x} in the training set has its own rectangle. But that is not helpful in a practical sense, in that it corresponds to a very "low bias/high variance"/complex predictor. So how does one find a tree of appropriate size? How does one choose a size at which to stop growing a tree, or more generally, prune a large tree back to a good size? (This latter is more general in that pruning a tree can produce subtrees not met in a sequence of trees as built up to a final one.) It turns out that it is possible to efficiently find a nested sequence of "optimal" subtrees of a large tree, and that methodology can in turn be used in cross-validation.

For T a subtree of some fixed large tree T_0 (e.g. grown until the cell with the fewest training \mathbf{x}_i contains 5 or less such points or in classification contexts until the training error is 0) write

$$E(T) = N \cdot \overline{\text{err}} = \sum_{i=1}^N L(\hat{f}(\mathbf{x}_i), y_i)$$

²⁸Much as we noted regarding nearest neighbor methods in Section 1.3.3, it can in some contexts be more useful to have the \widehat{p}_{mk} values themselves than to have only the 0-1 loss classifier derived from them.

(the total training error for the tree predictor based on T). For $\alpha > 0$ define the quantity

$$C_\alpha(T) = |T| + \alpha \cdot E(T)$$

(for, in the obvious way, $|T|$ the number of final nodes in the candidate tree).²⁹ Write

$$T(\alpha) = \underset{\text{subtrees } T}{\arg \min} C_\alpha(T)$$

and let \hat{f}_α be the corresponding predictor.

The question of how to find a subtree $T(\alpha)$ optimizing $C_\alpha(T)$ without making an exhaustive search over subtrees for every different value of α has a workable answer. There is a relatively small number of nested candidate subtrees that are the only ones that are possible minimizers of $C_\alpha(T)$, and as α decreases one moves through that nested sequence of subtrees from the largest/original tree to the smallest.

One may quickly search over all "pruned" versions of T_0 (subtrees T created by removing a node where there is a fork and all branches that follow below it) and find the one with minimum

$$\frac{E(T) - E(T_0)}{|T_0| - |T|}$$

(This IS the per node-of the lopped off branch of the first tree-increase in E .) Call that subtree T_1 . T_0 is the optimizer of $C_\alpha(T)$ over subtrees of T_0 for every $\alpha \geq (|T_0| - |T_1|) / (E(T_1) - E(T_0))$, but at $\alpha = (|T_0| - |T_1|) / (E(T_1) - E(T_0))$, the optimizing subtree switches to T_1 .

One then may search over all "pruned" versions of T_1 for the one with minimum

$$\frac{E(T) - E(T_1)}{|T_1| - |T|}$$

and call it T_2 . T_1 is the optimizer of $C_\alpha(T)$ over subtrees of T_0 for every $(|T_1| - |T_2|) / (E(T_2) - E(T_1)) \leq \alpha \leq (|T_0| - |T_1|) / (E(T_1) - E(T_0))$, but at $\alpha = (|T_1| - |T_2|) / (E(T_2) - E(T_1))$ the optimizing subtree switches to T_2 , and so on. (In this process, if there ever happens to be a tie among subtrees in terms of a minimizing a ratio of increase in total training error per decrease in number of nodes, one chooses the subtree with the smaller $|T|$.) For $T(\alpha)$ optimizing $C_\alpha(T)$, the function of α ,

$$C_\alpha(T(\alpha)) = \min_T C_\alpha(T)$$

is piecewise linear in α , and both it and the optimizing nested sequence of subtrees can be computed very efficiently in this fashion. Figure 29 illustrates the geometry of the situation. Notice that α is a complexity parameter and $|T(\alpha)|$ is non-decreasing in α .

²⁹It is, of course, equivalent to consider a quantity $E(T) + \lambda|T|$ for a $\lambda > 0$ in notation more like that used in other contexts like the ridge regression problem. Using α as a weight on $E(T)$ to define C_α , is equivalent to using $\lambda = 1/\alpha$ as a weight on $|T|$. The penalized form $E(T) + \lambda|T|$ is probably more often used (at least for user interface) in software implementations. The present C_α is more natural in the context of the development of optimal subtrees.

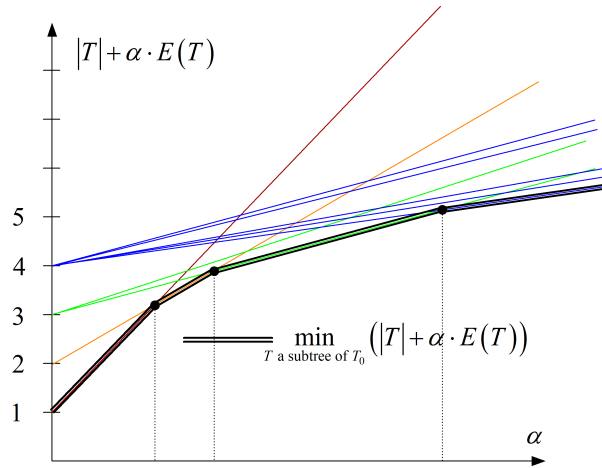


Figure 29: Cartoon of functions of α , $C_\alpha(T)$ for fixed T , and the optimized version $C_\alpha(T(\alpha))$.

One can then employ K -fold cross-validation to choose α as follows. For each of the K remainders $\mathbf{T} - \mathbf{T}_k$ (in the notation of Section 1.3.6)

1. grow an appropriate large tree (on a given dataset), then
2. "prune" the tree in 1. back by for each $\alpha > 0$ (a complexity parameter, weighting the remainder-in-training-sample error total E_k (for $\mathbf{T} - \mathbf{T}_k$) against complexity defined in terms of tree size) minimizing over choices of subtrees, the quantity

$$C_\alpha^k(T) = |T| + \alpha \cdot E_k(T)$$

(for $E_k(T)$ the error total for the corresponding tree predictor). Write

$$T_k(\alpha) = \arg \min_{\text{subtrees } T} C_\alpha^k(T)$$

and let \hat{f}_α^k be the corresponding predictor.

Then (as in Section 1.3.6), letting $k(i)$ be the index of the fold \mathbf{T}_k containing training case i , one computes the cross-validation error

$$CV(\alpha) = \frac{1}{N} \sum_{i=1}^N L(\hat{f}_\alpha^{k(i)}(\mathbf{x}_i), y_i)$$

For $\hat{\alpha}$ a minimizer of $CV(\alpha)$, one then operates on the entire training set, growing a large tree T and then finding the subtree, say $T(\hat{\alpha})$, optimizing $C_{\hat{\alpha}}(T) = |T| + \hat{\alpha} \cdot E(T)$, and using the corresponding predictor $\hat{f}_{\hat{\alpha}}$.

9.1.4 Measuring the Importance of Inputs for Tree Predictors

Consider the matter of assigning measures of "importance" of input variables for a tree predictor. In the spirit of ordinary linear models assessment of the importance of a predictor in terms of some reduction it provides in some error sum of squares, Breiman suggested the following. Suppose that in a regression or classification tree, input variable x_j provides the rectangle splitting criterion for nodes $node_{1j}, \dots, node_{m(j)j}$ and that before splitting at $node_{lj}$, the relevant rectangle R_{lj} has (for \hat{y}_{lj} the prediction fit for that rectangle) associated sum of training losses

$$E_{lj} = \sum_{i \text{ with } \mathbf{x}_i \in R_{lj}} L(\hat{y}_{lj}, y_i)$$

and that after splitting R_{lj} on variable x_j to create rectangles R_{lj}^1 and R_{lj}^2 (with respective fitted predictions \hat{y}_{lj}^1 and \hat{y}_{lj}^2) one has sums of training losses associated with those two rectangles

$$E_{lj}^1 = \sum_{i \text{ with } \mathbf{x}_i \in R_{lj}^1} L(\hat{y}_{lj}^1, y_i) \quad \text{and} \quad E_{lj}^2 = \sum_{i \text{ with } \mathbf{x}_i \in R_{lj}^2} L(\hat{y}_{lj}^2, y_i)$$

The reduction in total error provided by the split on x_j at $node_{lj}$ is thus

$$D_{lj} = E_{lj} - (E_{lj}^1 + E_{lj}^2)$$

(In regression/SEL contexts, this is a reduction in error sum of squares provided by the split of R_{lj} . In 0-1 loss classification contexts it is a reduction in training set misclassification errors.) One might then take

$$I_j = \sum_{l=1}^{m(j)} D_{lj}$$

a measure of the importance of x_j in fitting the tree and compare the various I_j s (or perhaps the square roots, $\sqrt{I_j}$ s).

Further, if a predictor is a (weighted) sum of regression trees (e.g. produced by "boosting" or in a "random forest") and I_{jm} measures the importance of x_j in the m th tree, then

$$I_j = \frac{1}{M} \sum_{m=1}^M I_{jm}$$

is perhaps one measure of the importance of x_j in the overall predictor. One can then compare the various I_j (or square roots) as a means of comparing the importance of the input variables.

9.2 PRIM (Patient Rule Induction Method)

This is another rectangle-based method of making a predictor on \mathfrak{R}^p . The language seems to be "patient" as opposed to "rash" and "rule induction" as in

"predictor development" or perhaps "conjunctive rule development" from the context of "market basket analysis." See Section 17.1 in regard to this latter usage.

PRIM can be thought of as a type of "bump-hunting." For a series of rectangles (or boxes) in p -space

$$R_1, R_2, \dots, R_l$$

one defines a predictor

$$\hat{f}_l(\mathbf{x}) = \begin{cases} \bar{y}_{R_1} & \text{if } \mathbf{x} \in R_1 \\ \bar{y}_{R_2 - R_1} & \text{if } \mathbf{x} \in R_2 - R_1 \\ \vdots & \vdots \\ \bar{y}_{R_m - \cup_{k=1}^{m-1} R_k} & \text{if } \mathbf{x} \in R_m - \cup_{k=1}^{m-1} R_k \\ \vdots & \vdots \\ \bar{y}_{(\cup_{k=1}^l R_k)^c} & \text{if } \mathbf{x} \notin \cup_{k=1}^l R_k \end{cases}$$

The boxes or rectangles are defined recursively in a way intended to catch "the remaining part of the input space with the largest output values." That is, to find R_1

1. identify a rectangle

$$\begin{aligned} l_1 &\leq x_1 \leq u_1 \\ l_2 &\leq x_2 \leq u_2 \\ &\vdots \\ l_p &\leq x_p \leq u_p \end{aligned}$$

that includes all input vectors in the training set,

2. identify a dimension, j , and either l_j or u_j so that by reducing u_j or increasing l_j just enough to remove a fraction α (say $\alpha = .1$) of the training vectors currently in the rectangle, the largest value of

$$\bar{y}_{\text{rectangle}}$$

possible is produced, and update that boundary of the rectangle,

3. repeat 2. until some minimum number of training inputs \mathbf{x}_i remain in the rectangle (say, at least 10),
4. expand the rectangle in any direction (increase a u_j or decrease an l_j) adding a training input vector that provides a maximal increase in $\bar{y}_{\text{rectangle}}$, and
5. repeat 4. until no increase is possible by adding a single training input vector.

This produces R_1 . For what it is worth, step 2. is called "peeling" and step 4. is called "pasting."

Upon producing R_1 , one removes from consideration all training vectors with $\mathbf{x}_i \in R_1$ and repeats 1. through 5. to produce R_2 . This continues until a desired number of rectangles has been created. One may pick an appropriate number of rectangles (l is a complexity parameter) by cross-validation and then apply the procedure to the whole training set to produce a set of rectangles and predictor on p -space that is piece-wise constant on regions built from boolean operations on rectangles.

PRIM is not anywhere near as common as classification and regression trees, but shares with them some of their attractive features, especially invariance to monotone transformation of coordinates of an input vector.

10 Predictors Built on Bootstrap Samples

10.1 Bagging in General

One might make B bootstrap samples of N (random samples with replacement of size N) from the training set \mathbf{T} , say $\mathbf{T}_1^*, \mathbf{T}_2^*, \dots, \mathbf{T}_B^*$, and train on these bootstrap samples using a particular method of prediction to produce, say,

$$\text{predictor } \hat{f}^{*b} \text{ based on } \mathbf{T}_b^*$$

Rather than using these to estimate the prediction error as in Section 16.4, consider using them to build a predictor.

The possibility considered in Section 8.7 of HTF is the use of **bootstrap aggregation**, or "bagging" under SEL. This is use of the predictor

$$\hat{f}_{\text{bag}}(\mathbf{x}) \equiv \frac{1}{B} \sum_{b=1}^B \hat{f}^{*b}(\mathbf{x})$$

Notice that even for fixed training set \mathbf{T} and input \mathbf{x} , this is random (varying with the selection of the bootstrap samples). One might let E^* denote averaging over the creation of a single bootstrap sample and \hat{f}^* be the predictor derived from such a bootstrap sample and think of

$$E^* \hat{f}^*(\mathbf{x})$$

as the "true" bagging predictor under SEL (that has the simulation-based approximation $\hat{f}_{\text{bag}}(\mathbf{x})$). One is counting on a law of large numbers to conclude that $\hat{f}_{\text{bag}}(\mathbf{x}) \rightarrow E^* \hat{f}^*(\mathbf{x})$ as $B \rightarrow \infty$. Note too, that unless the operations applied to a training set to produce \hat{f} are linear, $E^* \hat{f}^*(\mathbf{x})$ will differ from the predictor computed from the training data, $\hat{f}(\mathbf{x})$. The primary motivation for SEL bagging is the hope of averaging (not-perfectly-correlated as they are built on not-completely-overlapping bootstrap samples) low-bias/high-variance predictors to reduce variance (while maintaining low bias).

A bagged predictor in the 0-1 loss classification case is

$$\hat{f}_{\text{bag}}^*(\mathbf{x}) = \arg \max_k \sum_{b=1}^B I[\hat{f}^{*b}(\mathbf{x}) = k]$$

(a majority vote combination of the individual classifiers). One here expects that for each k a law of large numbers will imply that

$$\frac{1}{B} \sum_{b=1}^B I[\hat{f}^{*b}(\mathbf{x}) = k] \rightarrow P^*[\hat{f}^*(\mathbf{x}) = k] \text{ as } B \rightarrow \infty$$

so that there is a limiting classifier

$$\arg \max_k P^*[\hat{f}^*(\mathbf{x}) = k]$$

for which $\hat{f}_{\text{bag}}^*(\mathbf{x})$ is a simulation-based approximation.

It is common practice to make a kind of running cross-validation estimate of error based on "out-of-bag" (OOB) samples as one builds a bagged predictor. Note that (in cases where all training cases are different) for large N on average \mathbf{T}_b^* fails to contain about 37% of training cases.³⁰ Then, for each b suppose one keeps track of the set of (OOB) indices $I(b) \subset \{1, 2, \dots, N\}$ for which the corresponding training vector does not get included in the bootstrap training set \mathbf{T}_b^* . In SEL contexts let

$$\hat{y}_{iB}^* = \frac{1}{\# \text{ of indices } b \leq B \text{ such that } i \in I(b)} \sum_{b \leq B \text{ such that } i \in I(b)} \hat{f}^{*b}(\mathbf{x}_i)$$

and in 0-1 loss classification contexts let

$$\hat{y}_{iB}^* = \arg \max_k \sum_{b \leq B \text{ such that } i \in I(b)} I[\hat{f}^{*b}(\mathbf{x}_i) = k]$$

Then in SEL regression contexts, a running cross-validation type of estimate of Err is

$$\text{OOB}(B) = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_{iB}^*)^2$$

and a corresponding estimate for 0-1 loss classification contexts is

$$\text{OOB}(B) = \frac{1}{N} \sum_{i=1}^N I[y_i \neq \hat{y}_{iB}^*]$$

One then expects the convergence of $\text{OOB}(B)$, and plotting of $\text{OOB}(B)$ versus B is a standard way of trying to assess whether enough bootstrap samples have

³⁰The probability that a particular training case is missed in a bootstrap sample is $(1 - N^{-1})^N \approx e^{-1} \approx .37$ for N of any reasonable size.

been made to adequately represent the limiting predictor. In spite of the fact that for small B the (random) predictor \hat{f}_B^* is built on a small number of samples trees and is fairly simple, B is *not* really a complexity parameter, but is rather a *convergence* parameter.

Where losses other than SEL or 0-1 loss are involved, exactly how to "bag" bootstrapped versions of a predictor is not altogether obvious, and apparently even what might look like sensible possibilities can do poorly.

10.2 Random Forests: Special Bagging of Tree Predictors

This is an elaboration of the "bagging" (bootstrap aggregation) idea of Section 10.1 applied specifically to (regression and classification) trees. For each one of B bootstrap samples of N (from the training set \mathbf{T}), \mathbf{T}_b^* , develop a corresponding regression or classification tree by

1. at each node, randomly selecting m of the p input variables and finding an optimal single split of the corresponding rectangle over the selected input variables, splitting the rectangle, and
2. repeating 1 at each node up to a fixed depth or until no single-split improvement in splitting criterion is possible without creating a rectangle with less than a small number of training cases, n_{\min} .

(Note that no pruning is applied in this development.) Then let $\hat{f}^{*b}(\mathbf{x})$ be the corresponding tree-based predictor (taking values in \mathfrak{R} in the regression case or in $\mathcal{G} = \{1, 2, \dots, K\}$ in the classification case). A random forest predictor in the regression case is then

$$\hat{f}_B^*(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^B \hat{f}^{*b}(\mathbf{x})$$

and a 0-1 loss random classifier is

$$\hat{f}_B^*(\mathbf{x}) = \arg \max_k \sum_{b=1}^B I[\hat{f}^{*b}(\mathbf{x}) = k]$$

(This is a "majority vote" of the B constituent classification trees.)

As we have noted before in reference to nearest neighbor classification and classification trees, it can be more important in K -class classification models to estimate $P[y = k|\mathbf{x}]$ than it is to approximate the optimal classifier at \mathbf{x} . If, for tree b in a forest of B such trees,

$\mathcal{I}^b(\mathbf{x})$ = the set of indices of training cases with \mathbf{x}_i in the same rectangle as \mathbf{x}

then

$$\hat{p}_k^b(\mathbf{x}) = \frac{\sum_{\mathbf{x}_i \in \mathcal{I}^b(\mathbf{x})} I[y_i = k]}{\#\{\mathbf{x}_i \in \mathcal{I}^b(\mathbf{x})\}}$$

(the fraction of training cases with \mathbf{x}_i in the same rectangle as \mathbf{x} and $y_i = k$) estimates this probability using tree b . Then one random forest estimate of $P[y = k|\mathbf{x}]$ is the simple average

$$\frac{1}{B} \sum_{b=1}^B \hat{p}_k^b(\mathbf{x})$$

An alternative possibility is the weighted average

$$\frac{\sum_{b=1}^B \# [\mathbf{x}_i \in \mathcal{I}^b(\mathbf{x})] \cdot \hat{p}_k^b(\mathbf{x})}{\sum_{b=1}^B \# [\mathbf{x}_i \in \mathcal{I}^b(\mathbf{x})]} = \frac{\sum_{\mathbf{x}_i \in \mathcal{I}^b(\mathbf{x})} I[y_i = k]}{\sum_{b=1}^B \# [\mathbf{x}_i \in \mathcal{I}^b(\mathbf{x})]}$$

The basic tuning parameters in the development of $\hat{f}_B^*(\mathbf{x})$ are then m , and n_{\min} , and (if used) a maximum tree depth. Standard default values of parameters are

- $m = \lfloor p/3 \rfloor$ and $n_{\min} = 5$ for regression problems, and
- $m = \lfloor \sqrt{p} \rfloor$ and $n_{\min} = 1$ for classification problems.

The default $n_{\min} = 1$ for classification problems means that splitting terminates only because of reaching a maximum depth or the impossibility of reducing the splitting criterion with a single additional split. In the event that the maximum tree depth really doesn't come into play (because it is set to some value that is large in relative terms) this will produce random forest classifiers with 0 training error rate. (Any given training case will be missed by only about 37% of B bootstrap samples, so that about 63% of the B bootstrap samples will produce a tree correctly classifying the case, and so majority voting means that the random forest will correctly classify the case.) But notice that this does not imply that the out-of-bag-error $\text{OOB}(B)$ will be 0. And it does not imply that $\text{OOB}(B)$ for large B is unreliable as an indicator of the likely performance of a random forest classifier. It only implies that the training error rate is completely unreliable as an indicator of random forest classifier efficacy.

There is a fair amount of confusing discussion in the literature about the impossibility of a random forest "overfitting" with increasing B . This seems to be related to test error *not* initially-decreasing-but-then-increasing-in- B (which is perhaps loosely related to $\text{OOB}(B)$ converging to a positive value associated with the limiting predictor \hat{f}^{rf} (and not showing such behavior) and/or 0 training error rate for a random forest classifier not implying overfit³¹). But as HTF point out on their page 596, it is an entirely different question as to whether \hat{f}^{rf} itself is "too complex" to be adequately supported by the training data, \mathbf{T} . (And the whole discussion seems very odd in light of the fact that for any finite B , a different choice of bootstrap samples would produce a different \hat{f}_B^* as a new randomized approximation to \hat{f}^{rf} . Even for fixed \mathbf{x} , the value $\hat{f}_B^*(\mathbf{x})$ is a

³¹For many predictors/classifiers a 0 training error does suggest over-fit, but not necessarily for random forest classifiers.

random variable. Only $\hat{f}^{\text{rf}}(\mathbf{x})$ is fixed.) The fact that the out of bag error *will increase* if optimal allowable tree complexity (encoded in n_{min} and tree depth) and/or optimal m are exceeded means that a random forest $\hat{f}^{\text{rf}}(\mathbf{x})$ can indeed overfit (be too complex for the real information content of the training set).

There is also a fair amount of confusing discussion in the literature about the role of the *random* selection of the m predictors to use at each node-splitting (and the choice of m) in reducing "correlation between trees in the forest." The Breiman/Cutler web site http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm says that the "forest error rate" (presumably the error rate for \hat{f}^{rf}) depends upon "the correlation between any two trees in the forest" and the "strength of each tree in the forest." The meaning of "correlation" and "strength" is not clear if anything technical/precise is intended. One possibility for the first is some version of correlation between values of $\hat{f}^{*1}(\mathbf{x})$ and $\hat{f}^{*2}(\mathbf{x})$ as one repeatedly selects *the whole training set* \mathbf{T} in iid fashion from P and then makes two bootstrap samples—Section 15.4 of HTF seems to use this meaning.³² A meaning of the second is presumably some measure of average effectiveness of a single \hat{f}^{*b} . HTF Section 15.4 goes on to suggest that increasing m increases both "correlation" and "strength" of the trees, the first degrading error rate and the second improving it, and that the OOB estimate of error can be used to guide choice of m (usually in a broad range of values that are about equally attractive) if something besides the default is to be used.

10.3 Measuring the Importance of Inputs for Bagged Predictors

An idea of Breiman (phrased originally for random forests, but relevant to any bagged predictor) is this. For every bootstrap sample \mathbf{T}_b^* and predictor \hat{f}^{*b} based on the corresponding remainder $\mathbf{T} - \mathbf{T}_b^*$, one can compute a b th average error across the corresponding OOB sample, say

$$\overline{\text{err}}_b = \frac{1}{\# \left[\begin{array}{l} i \\ \text{case } i \text{ is not in the} \\ \text{bootstrap sample } b \end{array} \right]} \sum_{\substack{i \text{ s.t. case } i \text{ is not in the} \\ \text{the bootstrap sample } b}} L \left(\hat{f}^{*b}(\mathbf{x}_i), y_i \right)$$

Then in the OOB sample randomly permute the values of the j th coordinate of the input vectors, producing, say, input vectors $\tilde{\mathbf{x}}_i^j$. One can then define

$$\widetilde{\text{err}}_b^j = \frac{1}{\# \left[\begin{array}{l} i \\ \text{case } i \text{ is not in the} \\ \text{bootstrap sample } b \end{array} \right]} \sum_{\substack{i \text{ s.t. case } i \text{ is not in the} \\ \text{the bootstrap sample } b}} L \left(\hat{f}^{*b}(\tilde{\mathbf{x}}_i^j), y_i \right)$$

and take the difference

$$I_b^j = \widetilde{\text{err}}_b^j - \overline{\text{err}}_b \tag{112}$$

³²A second possibility concerns "bootstrap randomization distribution" correlation (for a fixed training set and a fixed \mathbf{x}) between values $\hat{f}^{*1}(\mathbf{x})$ and $\hat{f}^{*2}(\mathbf{x})$.

as an indicator (for the b th bootstrap sample) of the importance of variable j to prediction. These can then be averaged across the B bootstrap samples to produce

$$I^j = \frac{1}{B} \sum_{b=1}^B I_b^j \quad (113)$$

as a variable importance measure for variable j , and compared across j . (Typically these will be positive and large values are indicative of high variable importance.)

When applied to its specially constructed trees, these ideas produce a variable importance measure for a random forest. It is worth saying that what is made is then something different than what was suggested at the end of Section 9.1.4 for a predictor that is ultimately an average of tree predictors (that could also be employed for the random forest).

10.3.1 The Boruta Wrapper/Heuristic for Variable Selection

A methodology of Kurasa and Rudnicki for identification of all coordinates of an input that have "statistically detectable" variable importance builds on the importance measure I^j in display (113), usually derived from random forests.³³ It is aimed at judging which I^j s are "clearly more than noise." To enable this, when r predictors are currently under consideration some (say $s \geq \max(5, r)$) additional "shadow" (plausible noise) predictors are considered along with the actual predictors. These shadow predictors are made by randomly permuting entries in columns of the original input matrix for the predictors under consideration. These "should" prove to be of no importance in the prediction of y .

Boruta operates in stages in a "backwards elimination" fashion, beginning with consideration of all p original predictors and at a given stage dropping from the set of remaining potentially important variables those that are "clearly no better" than the best shadow variable at the stage. What is done to make decisions about elimination is to consider the set of values I_b^j defined in display (112) for a given j (newly indexing both those actual predictors still under consideration as $1, \dots, r$ and those shadow predictors newly generated at the beginning of the stage as $r + 1, \dots, r + s$), and compute both their mean I^j (in expression (113)) and their sample standard deviation, call it S^j . Some kind of rough test of "statistical significance" based on comparison of the scores (possibly accumulated across stages)

$$Z^j \equiv \frac{I^j}{S^j}$$

for real inputs (i.e. for $j = 1, \dots, r$) against

$$\max_{j=r+1, \dots, r+s} Z^j$$

³³Boruta is the name of the mythological Slavik god of the forest.

The elimination process is intended to ultimately drop from consideration all those predictors whose scores are not clearly bigger than those of (by construction useless) shadow predictors.

This is, of course, a heuristic and exact details vary with implementation. But the central idea is above and makes sense. It can be applied to any bagging context, and variants of it could be applied where one is not bagging, but other forms of holding out a test set are employed. Typically, the prediction method used is the random forest, because of its reputation for broad effectiveness and its independence of scaling of coordinates of the input. But there is nothing preventing its use with, say, a linear prediction or smoothing methodology.

10.4 Bumping and "Active Set Selection"

Another/different thing one might do with bootstrap versions of a predictor is to "pick-a-winner" based on performance on the training data. This is the "bumping"/stochastic perturbation idea of HTF's Section 8.9. That is, let $\hat{f}^{*0} = \hat{f}$ be the predictor computed from the training data, and define

$$\hat{b} = \arg \min_{b=0,1,\dots,B} \sum_{i=1}^N \left(y_i - \hat{f}^{*b}(\mathbf{x}_i) \right)^2$$

and take

$$\hat{f}_{\text{bump}}(\mathbf{x}) = \hat{f}^{*\hat{b}}(\mathbf{x})$$

The idea here is that if a few cases in the training data are responsible for making a basically good method of predictor construction perform poorly, eventually a bootstrap sample will miss those cases and produce an effective predictor.

Rick (Wen) Zhou in his ISU PhD dissertation made another use of bootstrapping, motivated by a real 2-class classification problem with "covariate shift." \mathbf{x} values in an important test set were mostly unlike input vectors \mathbf{x}_i available in a fairly small training set. With relatively little information available in the training set, highly flexible methods like nearest neighbor classification seemed unlikely to be effective. But a single simple application of a less flexible methodology (like one based on logistic regression) also seemed unlikely to be effective, because most test case input vectors were "near" at most "a few" training case input vectors and extrapolation of some kind was unavoidable.

What Zhou settled on and ultimately found to be relatively effective was to use (locally defined) bootstrap classifiers based on weighted bootstrap samples, with weights chosen to depend upon \mathbf{x} at which one is classifying. For a test input vector $\mathbf{x} \in \mathfrak{R}^p$ define weights for training case inputs \mathbf{x}_i by

$$w_i(\mathbf{x}) = \exp\left(-\eta \|\mathbf{x} - \mathbf{x}_i\|^2\right)$$

for some appropriate $\eta > 0$. For $w(\mathbf{x}) = \sum_{i=1}^N w_i(\mathbf{x})$ a single "weighted bootstrap" sample tailored to the input \mathbf{x} can be made by sampling N training cases iid according to the distribution over $i = 1, 2, \dots, N$ with probabilities

$p_i(\mathbf{x}) = w_i(\mathbf{x})/w(\mathbf{x})$. Upon fitting a simple form of classifier to B such tailored samples and using majority voting of those classifiers, one has a classification decision for input \mathbf{x} . It is one that respects both the likelihood that training cases close to the input are most relevant to decisions about its likely response and the need to enforce simplicity on the prediction.

11 "Ensembles" of Predictors

Bagging combines an "ensemble" of predictors consisting of versions of a single predictor computed from different bootstrap samples. An alternative might be to somehow weight together (or otherwise combine) different predictors (potentially even based on different models or methods). Here we consider 3 versions of this basic idea of somehow combining an ensemble of predictors to produce one better than any element of the ensemble.

11.1 Bayesian Model Averaging for Prediction

One theoretically straightforward way to justify this kind of enterprise is through the Bayes "multiple model" scenario (also used in Section 16.2.2). Suppose that M models P_1, P_2, \dots, P_M for (\mathbf{x}, y) are under consideration, the m th of which has parameter vector $\boldsymbol{\theta}_m$ and corresponding density $p_m(\mathbf{x}, y|\boldsymbol{\theta}_m)$. Then for the m th model (repeatedly abusing notation by using p to name many different functions) the training set \mathbf{T} has density

$$p_m(\mathbf{T}|\boldsymbol{\theta}_m) = \prod_{i=1}^N p_m(\mathbf{x}_i, y_i|\boldsymbol{\theta}_m)$$

We'll suppose here that $\boldsymbol{\theta}_m$ is not known and that it has prior density $g_m(\boldsymbol{\theta}_m)$ (for the m th model) and that and a prior probability for model m is

$$\pi(m)$$

Then a joint distribution for $m, \boldsymbol{\theta}_m, \mathbf{T}$, and (\mathbf{x}, y) has density

$$p_m(\mathbf{x}, y|\boldsymbol{\theta}_m) p_m(\mathbf{T}|\boldsymbol{\theta}_m) g_m(\boldsymbol{\theta}_m) \pi(m)$$

This has a marginal density for $y, \mathbf{x}, \mathbf{T}$ that is

$$\sum_{m=1}^M \pi(m) \int p_m(\mathbf{x}, y|\boldsymbol{\theta}_m) p_m(\mathbf{T}|\boldsymbol{\theta}_m) g_m(\boldsymbol{\theta}_m) d\boldsymbol{\theta}_m$$

from which the conditional mean of $y|\mathbf{x}, \mathbf{T}$ is

$$\mathbb{E}[y|\mathbf{x}, \mathbf{T}] = \frac{\sum_{m=1}^M \pi(m) \int \int y p_m(\mathbf{x}, y|\boldsymbol{\theta}_m) p_m(\mathbf{T}|\boldsymbol{\theta}_m) g_m(\boldsymbol{\theta}_m) d\boldsymbol{\theta}_m dy}{\sum_{m=1}^M \pi(m) \int \int p_m(\mathbf{x}, y|\boldsymbol{\theta}_m) p_m(\mathbf{T}|\boldsymbol{\theta}_m) g_m(\boldsymbol{\theta}_m) d\boldsymbol{\theta}_m dy}$$

Given m (the identity of the "correct" model) the variables \mathbf{T} , $\boldsymbol{\theta}_m$, and (\mathbf{x}, y) have joint density

$$p_m(\mathbf{x}, y|\boldsymbol{\theta}_m) p_m(\mathbf{T}|\boldsymbol{\theta}_m) g_m(\boldsymbol{\theta}_m)$$

for which the conditional mean of $y|\mathbf{x}, \mathbf{T}, m$ is, say,

$$E[y|\mathbf{x}, \mathbf{T}, m] = \frac{\int \int y p_m(\mathbf{x}, y|\boldsymbol{\theta}_m) p_m(\mathbf{T}|\boldsymbol{\theta}_m) g_m(\boldsymbol{\theta}_m) d\boldsymbol{\theta}_m dy}{\int \int p_m(\mathbf{x}, y|\boldsymbol{\theta}_m) p_m(\mathbf{T}|\boldsymbol{\theta}_m) g_m(\boldsymbol{\theta}_m) d\boldsymbol{\theta}_m dy}$$

so that

$$\begin{aligned} & \int \int y p_m(\mathbf{x}, y|\boldsymbol{\theta}_m) p_m(\mathbf{T}|\boldsymbol{\theta}_m) g_m(\boldsymbol{\theta}_m) d\boldsymbol{\theta}_m dy \\ &= E[y|\mathbf{x}, \mathbf{T}, m] \cdot \int \int p_m(\mathbf{x}, y|\boldsymbol{\theta}_m) p_m(\mathbf{T}|\boldsymbol{\theta}_m) g_m(\boldsymbol{\theta}_m) d\boldsymbol{\theta}_m dy \end{aligned}$$

from whence

$$E[y|\mathbf{x}, \mathbf{T}] = \frac{\sum_{m=1}^M E[y|\mathbf{x}, \mathbf{T}, m] \pi(m) \int \int p_m(\mathbf{x}, y|\boldsymbol{\theta}_m) p_m(\mathbf{T}|\boldsymbol{\theta}_m) g_m(\boldsymbol{\theta}_m) d\boldsymbol{\theta}_m dy}{\sum_{m=1}^M \pi(m) \int \int p_m(\mathbf{x}, y|\boldsymbol{\theta}_m) p_m(\mathbf{T}|\boldsymbol{\theta}_m) g_m(\boldsymbol{\theta}_m) d\boldsymbol{\theta}_m dy}$$

This is the average of $E[y|\mathbf{x}, \mathbf{T}, m]$ with respect to the conditional distribution (the "posterior" distribution) of $m|\mathbf{x}, \mathbf{T}$ specified by

$$\pi(m|\mathbf{x}, \mathbf{T}) = \frac{\pi(m) \int \int p_m(\mathbf{x}, y|\boldsymbol{\theta}_m) p_m(\mathbf{T}|\boldsymbol{\theta}_m) g_m(\boldsymbol{\theta}_m) d\boldsymbol{\theta}_m dy}{\sum_{m=1}^M \pi(m) \int \int p_m(\mathbf{x}, y|\boldsymbol{\theta}_m) p_m(\mathbf{T}|\boldsymbol{\theta}_m) g_m(\boldsymbol{\theta}_m) d\boldsymbol{\theta}_m dy}$$

That is, optimal SEL prediction of y proceeds by weighting what would be optimal predictors of y from the M constituent models by the relevant (updated from $\pi(m)$ by the information in \mathbf{x} and \mathbf{T} about the relevant density $p_m(\mathbf{x}, y|\boldsymbol{\theta}_m)$) conditional probabilities of the M components. This is "Bayes model averaging."

Essentially the same argument pertains in cases where y takes values in $\mathcal{G} = \{1, 2, \dots, K\}$ and 0-1 loss is involved. Under the same model as above, $P[y = k|\mathbf{x}, \mathbf{T}]$ is a $\pi(m|\mathbf{x}, \mathbf{T})$ -weighted average of $P[y = k|\mathbf{x}, \mathbf{T}, m]$ s appropriate under the M constituent models. (Of course, integrals "dy" are sums.) Ultimately, optimal 0-1 loss classifiers then choose for input \mathbf{x} (and training set \mathbf{T}) the class k maximizing this Bayes model average probability.

These developments of Bayes model averaging predictors explicitly involve \mathbf{x} in the posterior distribution of m (given \mathbf{x} and \mathbf{T}). This is because if one thinks of a new \mathbf{x} and corresponding y as generated by the same mechanism that produces \mathbf{T} , the observed \mathbf{x} is informative about m . Another way of modeling and calculating is the following.

One might suppose that the functions of \mathbf{x} ,

$$\mu_m(\mathbf{x}) = \frac{\int \int y p_m(\mathbf{x}, y|\boldsymbol{\theta}_m) g_m(\boldsymbol{\theta}_m) d\boldsymbol{\theta}_m dy}{\int \int p_m(\mathbf{x}, y|\boldsymbol{\theta}_m) g_m(\boldsymbol{\theta}_m) d\boldsymbol{\theta}_m dy}$$

or

$$p_m(y|\mathbf{x}) = \frac{\int p_m(\mathbf{x}, y|\boldsymbol{\theta}_m) g_m(\boldsymbol{\theta}_m) d\boldsymbol{\theta}_m}{\sum_{y=1}^K \int p_m(\mathbf{x}, y|\boldsymbol{\theta}_m) g_m(\boldsymbol{\theta}_m) d\boldsymbol{\theta}_m}$$

are objects of interest, but without a necessary connection to a specific new observation \mathbf{x} , itself informative about m and $\boldsymbol{\theta}_m$. (These functions are the conditional means and densities for y given \mathbf{x} under particular models m .) Positing a distribution specified by

$$p_m(\mathbf{T}|\boldsymbol{\theta}_m) g_m(\boldsymbol{\theta}_m) \pi(m)$$

for $m, \boldsymbol{\theta}_m, \mathbf{T}$ in the multiple model scenario, the posterior distribution for m given \mathbf{T} has pmf

$$\pi(m|\mathbf{T}) = \frac{\pi(m) \int p_m(\mathbf{T}|\boldsymbol{\theta}_m) g_m(\boldsymbol{\theta}_m) d\boldsymbol{\theta}_m}{\sum_{m=1}^M \pi(m) \int p_m(\mathbf{T}|\boldsymbol{\theta}_m) g_m(\boldsymbol{\theta}_m) d\boldsymbol{\theta}_m}$$

So the posterior mean of $\mu_m(\mathbf{x})$ given \mathbf{T} is

$$\sum_{m=1}^M \mu_m(\mathbf{x}) \pi(m|\mathbf{T})$$

and the posterior mean of $p_m(y|\mathbf{x})$ given \mathbf{T} is

$$\sum_{m=1}^M p_m(y|\mathbf{x}) \pi(m|\mathbf{T})$$

These differ from the previous "Bayes model averages," but they also represent sensible ensembles of predictors appropriate in the constituent models.

11.2 Stacking: SEL ... and 0-1 Loss

The Bayes model averaging idea is theoretically unimpeachable, but rarely practical. It does, however, raise the question "What might be suggested like this, but with a less Bayesian flavor?" One line of thinking is as follows.

Suppose that M SEL predictors are available (all based on the same training data), $\hat{f}_1, \hat{f}_2, \dots, \hat{f}_M$. One might seek a weight vector \mathbf{w} for which the predictor

$$\hat{f}(\mathbf{x}) = \sum_{m=1}^M w_m \hat{f}_m(\mathbf{x})$$

is effective. Why this can improve on any single one of the \hat{f}_m s is in some sense, this is "obvious." The set of possible \mathbf{w} (over which one searches for good weights) includes vectors with one entry 1 and all others 0. But to indicate in a concrete setting why this might work, consider a case where $M = 2$ and according to the $P^N \times P$ joint distribution of $(\mathbf{T}, (\mathbf{x}, y))$

$$E(y - \hat{f}_1(\mathbf{x})) = 0$$

and

$$\mathbb{E} \left(y - \hat{f}_2(\mathbf{x}) \right) = 0$$

Define

$$\hat{f}_\alpha = \alpha \hat{f}_1 + (1 - \alpha) \hat{f}_2$$

Then

$$\begin{aligned} \mathbb{E} \left(y - \hat{f}_\alpha(\mathbf{x}) \right)^2 &= \mathbb{E} \left(\alpha \left(y - \hat{f}_1(\mathbf{x}) \right) + (1 - \alpha) \left(y - \hat{f}_2(\mathbf{x}) \right) \right)^2 \\ &= \text{Var} \left(\alpha \left(y - \hat{f}_1(\mathbf{x}) \right) + (1 - \alpha) \left(y - \hat{f}_2(\mathbf{x}) \right) \right) \\ &= (\alpha, 1 - \alpha) \text{Cov} \begin{pmatrix} y - \hat{f}_1(\mathbf{x}) \\ y - \hat{f}_2(\mathbf{x}) \end{pmatrix} \begin{pmatrix} \alpha \\ 1 - \alpha \end{pmatrix} \end{aligned}$$

This is a quadratic function of α , that (since covariance matrices are non-negative definite) has a minimum. Thus there is a minimizing α that typically (is not 0 or 1 and thus) produces better expected loss than either $\hat{f}_1(\mathbf{x})$ or $\hat{f}_2(\mathbf{x})$.

More generally, again using the $P^N \times P$ joint distribution of $(\mathbf{T}, (\mathbf{x}, y))$, one may consider the random vector $(\hat{f}_1(\mathbf{x}), \hat{f}_2(\mathbf{x}), \dots, \hat{f}_M(\mathbf{x}), y)' = (\hat{\mathbf{f}}', y)$ and let

$$\mathbb{E} \begin{pmatrix} \hat{\mathbf{f}} \hat{\mathbf{f}}' \\ M \times M \end{pmatrix} \quad \text{and} \quad \mathbb{E} y \hat{\mathbf{f}} \begin{matrix} M \times 1 \end{matrix}$$

be respectively the matrix of expected products of the predictions and vector of expected products of y and elements of $\hat{\mathbf{f}}$. Upon writing out the expected square to be minimized and doing some matrix calculus, it's possible to see that optimal weights are of the form

$$\mathbf{w}^{\text{opt}} = \left(\mathbb{E} \left(\hat{\mathbf{f}} \hat{\mathbf{f}}' \right) \right)^{-1} \mathbb{E} y \hat{\mathbf{f}}$$

Of course, this isn't usable in practice, as the mean vector and expected cross product matrix are unknown.

One practical possibility is to "pick-a-winning" \mathbf{w} on the basis of LOO cross-validation. That is, for \hat{f}_m^i the m th predictor fit to the training set with the i th case removed,

$$\frac{1}{N} \sum_{i=1}^N \left(y_i - \left(w_0 + \sum_{m=1}^M w_m \hat{f}_m^i(\mathbf{x}_i) \right) \right)^2$$

is a LOOCVE for the predictor

$$\hat{f}(\mathbf{x}) = w_0 + \sum_{m=1}^M w_m \hat{f}_m(\mathbf{x})$$

that could be optimized as a function of $\mathbf{w} = (w_0, w_1, \dots, w_M)$ to produce $\mathbf{w}^{\text{stack}}$ and the "stacked" predictor

$$\hat{f}(\mathbf{x}) = w_0 + \sum_{m=1}^M w_m^{\text{stack}} \hat{f}_m(\mathbf{x})$$

An ad hoc version of stacking-type averaging is choice of weight vector \mathbf{w} based on informal consideration of one's (CV-supported) evaluation of the effectiveness of the individual predictors $\hat{f}_m(\mathbf{x})$ and the (training set) correlations between them. (Averaging multiple highly correlated predictors can't be expected to be particularly helpful, and individually effective predictors should get more weight than those that are relatively speaking ineffective.)

The application of the general notion of stacking to 0-1 loss classification has typically been treated on a very informal and ultimately unprincipled basis. Probably the most common suggestion extant in the machine learning world is to make classifications on a (potentially weighted) "majority vote" of an ensemble of classifiers. This is completely unsupported by any sensible theory. In this regard, see Vardeman and Morris "Majority Voting by Independent Classifiers can Increase Error Rates" that appears in *The American Statistician* in 2013 and their "Reply" to comments on the paper by Baker and others that appeared in the same journal in 2014.

A principled line of reasoning for the classification case is this. If a 0-1 loss classifier is any good, it is an approximation of the optimal form (28). So if it has an underlying voting function, that voting function must be equivalent to (must be a monotone transform of) an *approximate likelihood ratio*. What one is trying to do is find a better approximate likelihood ratio by combining several of these. It is then sensible to use underlying voting functions for the classifier (and the classifiers themselves in cases where no such voting function is available) **as features** input into a tree-based classification methodology (tree-based because of invariance to monotone transformation of coordinates of inputs and the fact that constituent voting functions are potentially on completely different scales, e.g. in some cases involving approximations for linear functions of $P[y = 1|\mathbf{x}]$ and in others approximations for $\mathcal{L}(\mathbf{x})$ directly). Details of sensible cross-validation to choose parameters of the constituent classification methods and the final tree-based method in this context remain to be considered. But the basic approach is clear and principled.

11.3 "Generalized Stacking" and "Deep" Structures for Prediction

Suppose that M predictors $\hat{f}_1, \hat{f}_2, \dots, \hat{f}_M$ (all based on the same training data) are available. We might call them together an "ensemble" of predictors and hope to make from them a single predictor that is more effective than any of the constituents. We have just said that for SEL prediction a linear combination of these (a "stacked" predictor) is one way of making such a predictor. We have also said that for 0-1 loss classification, combining multiple classifiers

through a tree-based function of their voting functions seems likely to be generally practically effective.³⁴ Here we consider the general problem "predictor combination." The primary contribution it potentially offers is **reduction of model bias** by adding flexibility not provided by any individual \hat{f}_m .

One important way to view the stacked SEL predictor

$$w_0 + \sum_{m=1}^M w_m \hat{f}_m(\mathbf{x}) \tag{114}$$

is as a linear predictor based on M new "features" that are the values of the ensemble. That suggests applying some standard predictor methodology to a "training set" consisting of M vectors of predictions ... *with or without some or all of the original input variables also reused as inputs*. The generalization of ordinary stacking is

$$\tilde{f}(\mathbf{x}) = \hat{f}\left(\hat{f}_1(\mathbf{x}), \hat{f}_2(\mathbf{x}), \dots, \hat{f}_M(\mathbf{x}), \mathbf{x}\right) \tag{115}$$

for some appropriate prediction algorithm \hat{f} . As this is more general than ordinary stacking, it has the potential to be even more effective than a linear combination of the M predictors could be in SEL problems and is applicable to other prediction problems.

(Generalized) Stacking is a big deal. From the earliest of the public predictive analytics contests (the Netflix Prize contest run 2006-2009) it has been common for winning predictions to be made by "end-of-game" merging of effort by two or more separate teams that in some way combine their separate predictions. More and more references are made on contest forums to various strategies for combining basic predictors. Multiple-level versions of the stacking structure are even discussed.³⁵

While the success of some (?luckiest among a number of?) ad hoc choices of generalized stacking forms in particular situations is undeniable, principled choices of forms and parameters for \hat{f} (and indeed $\hat{f}_1, \hat{f}_2, \dots, \hat{f}_M$) in display (115) involve both logical subtleties and huge computational demands. As always, cross-validation (or perhaps its OOB relative in the event that bagging is involved) is the only sound basis of these choices (and subsequent assessment of the implications of the choices).

Consider first a version of this problem where associated with each \hat{f}_m and with the top-level form \hat{f} are grids of possible values of parameters and a (potentially huge) **product grid** is searched for a best cross-validation error (and ultimately the optimizing parameter vector is applied to make the **pick-the-winner** meta-predictor (115)). For each (vector) element of the product grid, a cross-validation error is created by holding out folds and fitting \hat{f}_m s and \hat{f} with the prescribed parameter values on the remainders and testing on the corresponding folds. This is a perfectly defensible strategy for choosing a version

³⁴There is, unfortunately, a large and very confused "theoretical" literature on "classifier fusion" mostly built around the *ad hoc* notion of combination via majority voting.

³⁵In truth, they are but structured versions of the general form (115)

of predictor form (115). But notice that exactly as discussed in Section 1.3.7, the "winning" cross-validation error is not an honest indicator of the likely performance of the grid point/predictor ultimately chosen. In order to honestly estimate Err for the prediction methodology employed, one must cross-validate the whole process. In each of K remainders one would need to make grids and cross-validation errors for each grid point and pick a winner to predict on the corresponding fold in order to produce a cross-validation error for the pick-the-winner strategy. This implies a large computational load (especially if repeated cross-validation is done) in order to choose a final version of super-learner for application and assess the effectiveness of the process that produced it.

A second version of this scenario might pertain where ultimately individually-"optimized" (perhaps by cross-validation across some grid of parameter values for each m) versions of the \hat{f}_m s will be combined into a form (115) and choice of complexity parameters for \hat{f} then made by applying another subsequent "cross-validation," treating the chosen forms for the \hat{f}_m as fixed. The only way to assess the potential performance of this way of predicting is to do it (K times) on K folds and remainders. That is, within each of K remainders the whole sequence of choosing parameters for the \hat{f}_m s and subsequently for the \hat{f} must be repeated (by making K folds and remainders **within each remainder** ... surely leading to different "best" vectors of parameters for each fold) and applied to the corresponding fold to finally get a cross-validation error.

In both of these scenarios, it is clear that computation grows rapidly with the complexity of constituent predictor forms, the breath of the optimization desired, and the extent to which repetition of cross-validation is used.

What kind of top-level \hat{f} should be used in predictor form (115) could be investigated by comparison of cross-validation errors. The linear form (114) is most common and (at least in its ad hoc application) famously successful. But there is a very good case to be made that a random forest form has potential to be at least as effective in this role. Its invariance to scale of its inputs (inherited from its tree-based heritage) and wide success and reputation as an all-purpose tool make it a natural candidate.

Neural networks have the kind of "(potentially repeated) composition of multiple functions of the input vector" character evident in the form (115). That realization perhaps motivates consideration of versions of generalized stacking where the ensemble of predictors $\hat{f}_1, \hat{f}_2, \dots, \hat{f}_M$ itself has some specific kind of "neural-network-like" structure behind it. Figure 30 is a graphical representation of what is possible.

It is not at all obvious whether a neural-network-like structure for an ensemble of predictors in generalized stacking is necessarily helpful in practical prediction problems. The folklore in predictive analytics is that ordinary stacking is most helpful where elements of an ensemble have small correlations. (Obviously, if they are perfectly correlated no advantage can be gained by "combining" them.) How that folklore interacts with the current popularity of "deep learning" methods is unclear. One thing that *is* clear is that unthinking proliferation of "layers" in development of a predictor where they really add nothing

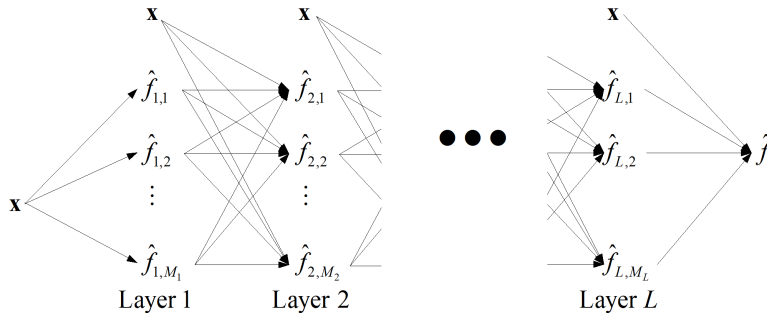


Figure 30: An L -layer structure for prediction based on \mathbf{x} .

to the empirical approximation of an optimal predictor can only exacerbate the computational problems of cross-validation and facilitate unwitting overfitting.

11.4 Boosting/Successive Approximation

11.4.1 SEL Boosting

A different line of thinking that leads to the use of weighted linear combinations of predictors is called **boosting**. The original classification version of the idea produces the famous "AdaBoost.M1" method discussed in Section 11.4.4. This methodology is really just an instance of the basic numerical analysis notion of **successive approximation** to find a solution to an equation or an optimizer of a functional.

There is general gradient boosting. But we begin with the SEL special case, because this version is both particularly easy to understand and explain and of high practical value. The basic idea is to repeatedly try to improve an approximator for $E[y|\mathbf{x}]$ by successively adding small corrections (based on modeling current residuals) to current approximators.

SEL boosting begins with some predictor $\hat{f}_0(\mathbf{x})$ (like, e.g., $\hat{f}_0(\mathbf{x}) = \bar{y}$). With an iterate $\hat{f}_{m-1}(\mathbf{x})$ in hand, one fits some SEL predictor, say $\hat{e}_m(\mathbf{x})$, to the N "data pairs" $(\mathbf{x}_i, y_i - \hat{f}_{m-1}(\mathbf{x}_i))$ consisting of inputs and current residuals. (Typically, some very simple/crude/non-complex "base predictor" form is used for \hat{e}_m .) Then, for some "learning rate" $\nu \in (0, 1)$, one sets

$$\hat{f}_m(\mathbf{x}) = \hat{f}_{m-1}(\mathbf{x}) + \nu \hat{e}_m(\mathbf{x})$$

One iterates on m through some number of iterations, M (possibly chosen by cross-validation). Commonly quoted choices for ν are numbers like .01 and the smaller is ν , the larger must be M . (Note that ν could be allowed to depend upon m , in which case notation like ν_m would be appropriate above.)

SEL boosting successively corrects a current predictor by adding to it some small fraction of a predictor for its residuals. The value ν functions as a com-

plexity or regularizing parameter, as does M . Small ν and large M correspond to large complexity. The boosting notion is different in spirit from stacking or model averaging, but like them ends with a linear combination of fitted forms as a final predictor/approximator for $E[y|\mathbf{x}]$.

This kind of sequential modification of a predictor is not discussed in ordinary regression/linear models courses because if a base predictor is an OLS predictor for a fixed linear model, corrections to an initial fit based on this same model fit to residuals will predict that all residuals are 0. In this circumstance boosting does nothing to change or improve an initial OLS fit.

11.4.2 General "Gradient Boosting"

Now consider approximate empirical optimization (over choice of real-valued function g) of

$$EL(g(\mathbf{x}), y)$$

through (successive approximation) search for predictor \hat{f} that optimizes

$$\sum_{i=1}^N L(\hat{f}(\mathbf{x}_i), y_i) = N \cdot \overline{\text{err}} \quad (116)$$

One begins with some predictor $\hat{f}_0(\mathbf{x})$ (like, e.g., $\hat{f}_0(\mathbf{x}) = \arg \min_{\hat{y}} \sum_{i=1}^N L(\hat{y}, y_i)$).

With an iterate $\hat{f}_{m-1}(\mathbf{x})$ in hand, one then might consider how to improve the current total training set loss $\sum_{i=1}^N L(\hat{f}_{m-1}(\mathbf{x}_i), y_i)$. Let

$$\tilde{y}_{im} = - \left. \frac{\partial}{\partial \hat{y}} L(\hat{y}, y_i) \right|_{\hat{y}=\hat{f}_{m-1}(\mathbf{x}_i)} \quad (117)$$

These values are the elements of the negative gradient of total loss with respect to the current predictions for the training set. Ideally, one would like to correct $\hat{f}_{m-1}(\mathbf{x})$ in a way that moves each prediction of a training output $\hat{f}_{m-1}(\mathbf{x}_i)$ by more or less a common multiple of \tilde{y}_{im} . To that end, one fits some SEL predictor, say $\hat{e}_m(\mathbf{x})$, to "data pairs" $(\mathbf{x}_i, \tilde{y}_{im})$. (As in the special case of SEL boosting, typically some very simple/crude/non-complex form of "base predictor" is used for \hat{e}_m .) Let $\rho_m > 0$ (controlling the "step-size" in modifying $\hat{f}_{m-1}(\mathbf{x})$) stand for a multiplier for $\hat{e}_m(\mathbf{x})$ such that

$$\sum_{i=1}^N L(\hat{f}_{m-1}(\mathbf{x}_i) + \rho_m \hat{e}_m(\mathbf{x}_i), y_i)$$

is small (ideally, minimum). (Unless an analytical formula for an optimal ρ_m is obvious, some kind of numerical line search is implicit in the good choice of ρ_m .)

Then, for some "learning rate" $\nu \in (0, 1)$, one sets

$$\hat{f}_m(\mathbf{x}) = \hat{f}_{m-1}(\mathbf{x}) + \nu \rho_m \hat{e}_m(\mathbf{x}) \quad (118)$$

as an approximate "steepest descent" correction. Of course, other criteria besides SEL (like AEL) could be used in fitting $\hat{e}_m(\mathbf{x})$ and ν could be allowed to change with m .

The development here allows for arbitrary base predictors. But for good reasons (especially the fact that trees are invariant to monotone transformations of coordinates of \mathbf{x}) the functions \hat{e}_m are often rectangle-based (and even restricted to single-split-trees in the case of AdaBoost.M1). If a tree-building algorithm for approximating the values (117) produces a set of non-overlapping rectangles R_1, R_2, \dots, R_L that cover the input space, rather than using for $\hat{e}_m(\mathbf{x})$ in rectangle R_l some average of the values \tilde{y}_{im} for training cases with $\mathbf{x}_i \in R_l$, it makes more sense to use

$$\hat{e}_m(\mathbf{x}) = \arg \min_c \sum_{i \text{ s.t. } \mathbf{x}_i \in R_l} L(\hat{f}_{m-1}(\mathbf{x}_i) + c, y_i) \quad \text{for } \mathbf{x} \in R_l \quad (119)$$

and $\rho_m = 1$ and this is the form typically used in gradient boosting with trees.

Update form (119) relies upon 1-dimensional optimizations of a sum of losses for training inputs in L tree-generated rectangles. Another way this idea can be used is with rectangles formed based on values of sub-vectors of \mathbf{x} with finite numbers of possible values. That is, consider again the context of Section 1.4.2. For a given choice of D categorical, ordinal, or finite-discrete coordinates of \mathbf{x} defining the sub-vector $\tilde{\mathbf{x}}$, consider using

$$\hat{e}_m(\mathbf{x}) = \arg \min_c \sum_{i \text{ s.t. } \tilde{\mathbf{x}}_i = \tilde{\mathbf{x}}} L(\hat{f}_{m-1}(\mathbf{x}_i) + c, y_i) \quad \text{for } \mathbf{x} \text{ with } \tilde{\mathbf{x}}_i = \tilde{\mathbf{x}}$$

and $\rho_m = 1$. This $\hat{e}_m(\mathbf{x})$ has only a finite number of possible values, one corresponding to each of the sets $\{i | \tilde{\mathbf{x}}_i = \tilde{\mathbf{x}}\}$. Further, in contexts where there are a number of potential choices of such sets of discrete coordinates of \mathbf{x} , the total losses after update (118) can be compared to choose a good sub-vector $\tilde{\mathbf{x}}$ to use to produce \hat{f}_m .

SEL We had a first look at SEL boosting in Section 11.4.1. To establish that it is a version of gradient boosting, simply suppose now that $L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$. Then

$$\tilde{y}_{im} = - \frac{\partial}{\partial \hat{y}} \left(\frac{1}{2} (\hat{y} - y_i)^2 \right) \Big|_{\hat{y} = \hat{f}_{m-1}(\mathbf{x}_i)} = y_i - \hat{f}_{m-1}(\mathbf{x}_i)$$

and for SEL the general gradient boosting corrections are indeed based on the prediction of ordinary residuals.

AEL (and Binary Regression Trees) Suppose now that $L(\hat{y}, y) = |\hat{y} - y|$. Then, beginning from $\hat{f}_0(\mathbf{x})$ (say $\hat{f}_0(\mathbf{x}) = \text{median}\{y_i\}$),

$$\tilde{y}_{im} = - \frac{\partial}{\partial \hat{y}} (|\hat{y} - y_i|) \Big|_{\hat{y} = \hat{f}_{m-1}(\mathbf{x}_i)} = \text{sign}(y_i - \hat{f}_{m-1}(\mathbf{x}_i))$$

So the gradient boosting update step is "fit a SEL predictor for ± 1 s coding the signs of the residuals from the previous iteration." In the event that the base predictors are regression trees, the $\hat{e}_m(\mathbf{x})$ in a rectangle will be a median of ± 1 s coming from signs of residuals for cases with \mathbf{x}_i in the rectangle (and thus have value either -1 or 1 , constant on the rectangle).

Standard Voting Functions for 2-Class Classification Referring again to the development in Section 1.5.3, recall that approximation to optimal voting functions $g(\mathbf{x})$ can produce approximately optimal 2-class classifiers $\text{sign}(g(\mathbf{x}))$. Then consider $h_1(u) = \ln(1 + \exp(-u)) / \ln(2)$ and the loss

$$L(g(\mathbf{x}), y) = h_1(yg(\mathbf{x})) = \ln(1 + \exp(-yg(\mathbf{x}))) / \ln(2)$$

For this situation

$$\tilde{y}_{im} = - \frac{\partial}{\partial \hat{y}} (\ln(1 + \exp(-y_i \hat{y})) / \ln(2)) \Big|_{\hat{y} = \hat{f}_{m-1}(\mathbf{x}_i)} = \frac{1}{\ln 2} \left(\frac{\hat{f}_{m-1}(\mathbf{x}_i) \exp(-y_i \hat{y}_i)}{1 + \exp(-y_i \hat{f}_{m-1}(\mathbf{x}_i))} \right)$$

and corresponding boosting can be expected to produce a voting function approximating

$$g^*(\mathbf{x}) = \ln \left(\frac{P[y = 1 | \mathbf{x}]}{P[y = -1 | \mathbf{x}]} \right)$$

For the exponential function $h_2(u) = \exp(-u)$ and loss $L(g(\mathbf{x}), y) = h_2(yg(\mathbf{x}))$ one has

$$\tilde{y}_{im} = - \frac{\partial}{\partial \hat{y}} \exp(-y_i \hat{y}) \Big|_{\hat{y} = \hat{f}_{m-1}(\mathbf{x}_i)} = y_i \exp(-y_i \hat{f}_{m-1}(\mathbf{x}_i))$$

and corresponding boosting produces a voting function approximating $\frac{1}{2}g^*(\mathbf{x})$ (for $g^*(\mathbf{x})$ above). (For the choice of base predictors as single-split trees, gradient boosting would be an approximate version of the famous AdaBoost.M1 algorithm.)

Finally, for the hinge function $h_3(u) = (1 - u)_+$ and loss $L(g(\mathbf{x}), y) = h_3(yg(\mathbf{x}))$, one gets

$$\tilde{y}_{im} = - \frac{\partial}{\partial \hat{y}} (1 - y_i \hat{y})_+ \Big|_{\hat{y} = \hat{f}_{m-1}(\mathbf{x}_i)} = y_i I[y_i \hat{f}_{m-1}(\mathbf{x}_i) < 1]$$

and corresponding boosting produces a voting function approximating the optimal classifier directly.

K -Class Classification Models We noted in Section 1.3.2 that in a K -class classification model, under the cross-entropy loss $L(\hat{\mathbf{y}}, y) = - \sum_{k=1}^K I[y = k] \ln(\hat{y}_k)$, for non-negative y_1, y_2, \dots, y_K summing to 1, predictors

$$f_k(\mathbf{x}) = P[y = k | \mathbf{x}]$$

are optimal and can be used to produce optimal 0-1 loss classifiers. Consider boosting to produce approximations to $f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_{K-1}(\mathbf{x})$. Begin with $K-1$ positive predictors $\hat{f}_{10}(\mathbf{x}), \hat{f}_{20}(\mathbf{x}), \dots, \hat{f}_{(K-1)0}(\mathbf{x})$ with $\sum_{k=1}^{K-1} \hat{f}_{k0}(\mathbf{x}) < 1$. (For example, $\hat{f}_{k0}(\mathbf{x}) = 1/K$ will serve.) Then for $\hat{y}_1, \hat{y}_2, \dots, \hat{y}_{K-1}$ positive with sum less than 1, with

$$L(\hat{\mathbf{y}}, y) = - \sum_{k=1}^{K-1} I[y = k] \ln(\hat{y}_k) - I[y = K] \ln\left(1 - \sum_{k=1}^{K-1} \hat{y}_k\right)$$

let (for $k = 1, 2, \dots, K-1$)

$$\begin{aligned} \tilde{y}_{ikm} &= - \left. \frac{\partial}{\partial \hat{y}_k} L(\hat{\mathbf{y}}, y_i) \right|_{\hat{\mathbf{y}}_k = \hat{f}_{m-1}(\mathbf{x}_i)} \\ &= I[y_i = k] \frac{1}{\hat{f}_{k(m-1)}(\mathbf{x}_i)} - I[y_i = K] \frac{1}{1 - \sum_{k=1}^{K-1} \hat{f}_{k(m-1)}(\mathbf{x}_i)} \end{aligned}$$

For each k fit some SEL predictor, say $\hat{e}_{km}(\mathbf{x})$, to pairs $(\mathbf{x}_i, \tilde{y}_{ikm})$ and for an appropriate $\nu_m > 0$ set

$$\hat{f}_{km}(\mathbf{x}) = \hat{f}_{k(m-1)}(\mathbf{x}) + \nu_m \hat{e}_{km}(\mathbf{x})$$

(ν_m will need to be chosen to be small enough that all $\hat{f}_{1m}(\mathbf{x}), \hat{f}_{2m}(\mathbf{x}), \dots, \hat{f}_{(K-1)m}(\mathbf{x})$ remain positive with sum less than 1.)

11.4.3 Some Issues Related to Boosting Practice

Here we consider several issues that arise in the use of boosting. These mostly concern control of complexity of predictors in boosting.

Where trees are used to create the functions \hat{e}_m , there is the question of how large they should be allowed to grow. The answer seems to be "Not too large, maybe to about 6 or so terminal nodes." Another (probably better) approach to this question would seem to be to grow large trees and then employ cost-complexity pruning, ultimately using cross-validation to choose a value for the weight α (or $\lambda = 1/\alpha$).

There is always question of the number of boosting steps, M , that should be employed. This can/should be limited in size (very large values surely producing overfit). Holding back a part of the training sample and watching performance of a predictor on that single test set as M increases is a possible crude method of choosing M . Presumably, cross-validation provides a more reliable means of directing choice of M .

"Shrinkage" also impacts final boosted predictor complexity. In choosing $\nu \in (0, 1)$ for use in update (118) one chooses a multiplier of $\hat{e}_m(\mathbf{x})$ strictly less than one that minimizes the updated total loss. That is, one doesn't make the "full correction" to \hat{f}_{m-1} in producing \hat{f}_m . The smaller is this parameter the larger will be M needed for good predictor performance. One might well choose both ν and M via cross-validation.

"Subsampling" or "stochastic boosting" is the practice of at each iteration of boosting, instead of choosing an update based on the whole training set, choosing a fraction η of the training set at random and fitting to it (using a new random selection at each update). This reduces computation time per iteration and can also improve predictor performance (primarily by reducing overfit?). Once more, cross-validation can inform the choice of η .

A very popular implementation of gradient boosting goes by the name "XGBoost" (for "eXtreme Gradient Boosting"). This is an R package (with similar implementations in other systems) that provides a lot of flexibility and code that is very fast to run (even providing parallelization where hardware supports it). The `caret` package can be used to do cross-validation based on XGBoost, allowing one to tune on a number of algorithm complexity parameters.

11.4.4 AdaBoost.M1

Consider a 2-class 0-1 loss classification problem with $-1/1$ coding of output y (y takes values in $\mathcal{G} = \{-1, 1\}$). The AdaBoost.M1 algorithm is an exact variant of the (approximate) gradient boosting algorithm, but is usually described in other terms. We describe those terms next, and then make the connection to general boosting.

The standard/original description of the AdaBoost.M1 algorithm is as follows.

1. Initialize weights on training data (\mathbf{x}_i, y_i) at

$$w_{i1} \equiv \frac{1}{N} \text{ for } i = 1, 2, \dots, N$$

2. Fit a \mathcal{G} -valued "stump" (single-split tree/single cut on a single coordinate of \mathbf{x}) predictor/classifier g_1 to the training data to optimize

$$\sum_{i=1}^N I[y_i \neq g(\mathbf{x}_i)]$$

let

$$\overline{\text{err}}_1 = \frac{1}{N} \sum_{i=1}^N I[y_i \neq g_1(\mathbf{x}_i)]$$

and define

$$\alpha_1 = \ln \left(\frac{1 - \overline{\text{err}}_1}{\overline{\text{err}}_1} \right)$$

3. Set new weights on the training data

$$w_{i2} = \frac{1}{N} \exp(\alpha_1 I[y_i \neq g_1(\mathbf{x}_i)]) \text{ for } i = 1, 2, \dots, N$$

(This up-weights misclassified observations by a factor of $(1 - \overline{\text{err}}_1) / \overline{\text{err}}_1$.)

4. For $m = 2, 3, \dots, M$

- (a) Fit a \mathcal{G} -valued stump predictor/classifier g_m to the training data to optimize

$$\sum_{i=1}^N w_{im} I[y_i \neq g(\mathbf{x}_i)]$$

- (b) Let

$$\overline{\text{err}}_m = \frac{\sum_{i=1}^N w_{im} I[y_i \neq g_m(\mathbf{x}_i)]}{\sum_{i=1}^N w_{im}}$$

- (c) Set

$$\alpha_m = \ln\left(\frac{1 - \overline{\text{err}}_m}{\overline{\text{err}}_m}\right)$$

- (d) Update weights as

$$\begin{aligned} w_{i(m+1)} &= w_{im} \exp(\alpha_m I[y_i \neq g_m(\mathbf{x}_i)]) \\ &= w_{im} \left(I[y_i = g_m(\mathbf{x}_i)] + \frac{1 - \overline{\text{err}}_m}{\overline{\text{err}}_m} I[y_i \neq g_m(\mathbf{x}_i)] \right) \end{aligned}$$

for $i = 1, 2, \dots, N$. (This up-weights misclassified observations by a factor of $(1 - \overline{\text{err}}_m)/\overline{\text{err}}_m$.)

5. Output a voting function

$$\sum_{m=1}^M \alpha_m g_m(\mathbf{x})$$

(based on "weighted voting" by the classifiers g_m) for an AdaBoost.M1 classifier

$$\hat{f}_M(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^M \alpha_m g_m(\mathbf{x})\right)$$

(Classifiers g_m with small $\overline{\text{err}}_m$ get big positive weights in the final "voting.")

Figure 31 is a graphic of a small ($N = 16$) fake $p = 2$ dataset and (single line) boundaries of $M = 7$ successive "stumps" used to develop an AdaBoost.M1 classifier with 0 training error rate. (Arrows point in the direction of $y = +1$ decisions.) Corresponding classifiers are portrayed in Figure 32.

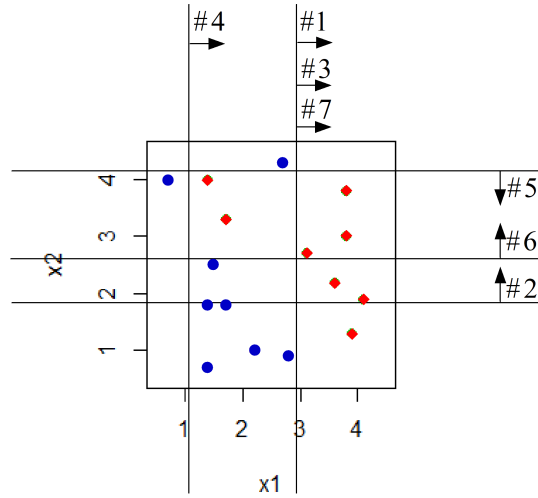


Figure 31: $M = 7$ consecutive AdaBoost.M1 cuts for a small fake data set.

AdaBoost.M1 as an Instance of General Boosting The AdaBoost.M1 algorithm is equivalent to an instance of general boosting for a voting function, based on the exponential loss function $h_2(v) \equiv \exp(-v)$ of Section 1.5.3. The argument is as follows. Take $g_1 = \frac{1}{2}\hat{f}_1$ for \hat{f}_1 as in the traditional description of AdaBoost.M1 to serve as an initial voting function to be improved through a series of boosting steps. Suppose that iterate g_{m-1} is in hand and one desires to improve (reduce) the total training loss

$$\sum_{i=1}^N \exp(-y_i g_{m-1}(\mathbf{x}_i))$$

by an update of voting function g_{m-1} to

$$g_m(\mathbf{x}) = g_{m-1}(\mathbf{x}) + \gamma_m \hat{e}_m(\mathbf{x}) \quad (120)$$

where \hat{e}_m is an appropriate stump classifier ("a single split tree" classifier) and γ_m is (without loss of generality) a positive constant.

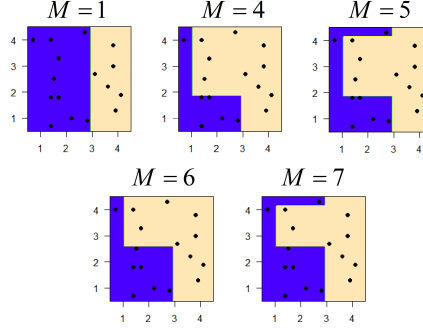


Figure 32: Classifiers corresponding to the voting functions from the cuts indicated in Figure 31.

The total training loss associated with the iterate (120) is

$$\begin{aligned}
& \sum_{i=1}^N \exp[-y_i (g_{m-1}(\mathbf{x}_i) + \gamma_m \hat{e}_m(\mathbf{x}_i))] \\
&= \sum_{i=1}^N \exp(-y_i g_{m-1}(\mathbf{x}_i)) \exp(-y_i \gamma_m \hat{e}_m(\mathbf{x}_i)) \\
&= \sum_{\substack{i \text{ s.t.} \\ y_i \neq \hat{e}_m(\mathbf{x}_i)}} \exp(-y_i g_{m-1}(\mathbf{x}_i)) \exp(\gamma_m) + \sum_{\substack{i \text{ s.t.} \\ y_i = \hat{e}_m(\mathbf{x}_i)}} \exp(-y_i g_{m-1}(\mathbf{x}_i)) \exp(-\gamma_m) \\
&= (\exp(\gamma_m) - \exp(-\gamma_m)) \sum_{\substack{i \text{ s.t.} \\ y_i \neq \hat{e}_m(\mathbf{x}_i)}} \exp(-y_i g_{m-1}(\mathbf{x}_i)) \\
&\quad + \exp(-\gamma_m) \sum_i \exp(-y_i g_{m-1}(\mathbf{x}_i))
\end{aligned}$$

So, whatever be the positive value of γ_m , $\hat{e}_m(\mathbf{x})$ should be chosen to minimize the 0-1 loss error rate for a single cut classifier for cases weighted proportional to values $\exp(-y_i g_{m-1}(\mathbf{x}_i))$.

Consider then choice of γ_m . The derivative of the total training loss with respect to γ_m is

$$\sum_{\substack{i \text{ s.t.} \\ y_i \neq \hat{e}_m(\mathbf{x}_i)}} \exp(-y_i g_{m-1}(\mathbf{x}_i)) \exp(\gamma_m) - \sum_{\substack{i \text{ s.t.} \\ y_i = \hat{e}_m(\mathbf{x}_i)}} \exp(-y_i g_{m-1}(\mathbf{x}_i)) \exp(-\gamma_m)$$

This is 0 when

$$\exp(2\gamma_m) = \frac{\sum_{\substack{i \text{ s.t.} \\ y_i = \hat{e}_m(\mathbf{x}_i)}} \exp(-y_i g_{m-1}(\mathbf{x}_i))}{\sum_{\substack{i \text{ s.t.} \\ y_i \neq \hat{e}_m(\mathbf{x}_i)}} \exp(-y_i g_{m-1}(\mathbf{x}_i))}$$

That is, an optimal γ_m is

$$\begin{aligned}\gamma_m &= \frac{1}{2} \ln \left(\frac{\sum_{\substack{i \text{ s.t.} \\ y_i = \hat{e}_m(\mathbf{x}_i)}} \exp(-y_i g_{m-1}(\mathbf{x}_i))}{\sum_{\substack{i \text{ s.t.} \\ y_i \neq \hat{e}_m(\mathbf{x}_i)}} \exp(-y_i g_{m-1}(\mathbf{x}_i))} \right) \\ &= \frac{1}{2} \ln \left(\frac{1 - r_m}{r_m} \right)\end{aligned}$$

for

$$r_m = \frac{\sum_{\substack{i \text{ s.t.} \\ y_i \neq \hat{e}_m(\mathbf{x}_i)}} \exp(-y_i g_{m-1}(\mathbf{x}_i))}{\sum_{i=1}^N \exp(-y_i g_{m-1}(\mathbf{x}_i))}$$

which is the 0-1 loss error rate for the classifier \hat{e}_m where weights on points in a training set are proportional to $\exp(-y_i g_{m-1}(\mathbf{x}_i))$.

Notice then that the ratios of the weights at stages $m - 1$ and m satisfy

$$\begin{aligned}& \frac{\exp(-y_i g_m(\mathbf{x}_i))}{\exp(-y_i g_{m-1}(\mathbf{x}_i))} \\ &= \frac{\exp(-y_i (g_{m-1}(\mathbf{x}_i) + \gamma_m \hat{e}_m(\mathbf{x}_i)))}{\exp(-y_i g_{m-1}(\mathbf{x}_i))} \\ &= \exp(-y_i \gamma_m \hat{e}_m(\mathbf{x}_i)) \\ &= \exp\left(-\frac{1}{2} \ln\left(\frac{1 - r_m}{r_m}\right)\right) I[\hat{e}_m(\mathbf{x}_i) = y_i] + \exp\left(\frac{1}{2} \ln\left(\frac{1 - r_m}{r_m}\right)\right) I[\hat{e}_m(\mathbf{x}_i) \neq y_i] \\ &= \left(\frac{1 - r_m}{r_m}\right)^{-1/2} I[\hat{e}_m(\mathbf{x}_i) = y_i] + \left(\frac{1 - r_m}{r_m}\right)^{1/2} I[\hat{e}_m(\mathbf{x}_i) \neq y_i] \\ &= \left(\frac{1 - r_m}{r_m}\right)^{-1/2} \left[I[\hat{e}_m(\mathbf{x}_i) = y_i] + \left(\frac{1 - r_m}{r_m}\right) I[\hat{e}_m(\mathbf{x}_i) \neq y_i] \right]\end{aligned}$$

Since r_m doesn't depend upon i , looking across i this is proportional to a ratio of 1 for cases with $I[\hat{e}_m(\mathbf{x}_i) = y_i]$ and ratio $(1 - r_m)/r_m$ for cases with $I[\hat{e}_m(\mathbf{x}_i) \neq y_i]$. That is (recalling the meaning of r_m) the ratios of weights for a given case in this development are completely equivalent to those produced by the updating prescribed in 4(d) of the standard description of AdaBoost.M1.

Ultimately then, all of this taken together establishes that this ("exact" as opposed to "gradient") boosting development produces an m th iterate of a voting function exactly half of that produced through m iterations of the standard development of AdaBoost.M1. Since the factor of $\frac{1}{2}$ is irrelevant to the sign of the voting function, the corresponding classifier is exactly the AdaBoost.M1 classifier.

11.5 Quinlan's Cubist and "Divide and Conquer" Strategies

There is a line of algorithms associated with Ross Quinlan (including "Cubist" and "C5.0," the former being a SEL prediction methodology and the latter a

classifier). His company web site is <https://www.rulequest.com/index.html>. His algorithms are very complicated, and complete descriptions do not seem to be publicly available. (Though there are open source versions of some of his algorithms, much of his work seems to be proprietary and commercial versions of his software are no doubt more reliable than the open source versions.) Text-book descriptions of his methods are generally vague. Probably the best ones I know of are in the KJ book.

The basic notion of Cubist seems to be to cut up an input space, \mathfrak{R}^p , into rectangles and fit a (different) linear predictor for y in each rectangle. Consider the rectangle

$$R = \{\mathbf{x} \mid a_1 < x_1 < b_1, a_2 < x_2 < b_2, \dots, a_p < x_p < b_p\}$$

where a_j and b_j can be finite or infinite. Where at least one of a_j or b_j is finite, a split on the input space has been made on coordinate j . Jargon typically used in describing these methods is that if one lists only rectangles where one or both of the a_j or b_j are finite, one has specified a "rule."

There are many implementation choices (the consequences of which are not transparent) that (much as with MARS) amount to a kind of "special sauce" owned by Quinlan and/or others who have followed him. Vague expositions of Cubist leave most users to treat SEL prediction based on it as a mysterious (albeit often effective) "black box."

Here are a few observations based on available information on Cubist for SEL prediction.

1. Trees of **regressions** (not trees with constant predictions in each final rectangle) seem to be at the heart of the methodology, both generating rectangles and making predictions. The "error" used to guide node splitting seems to be

$$\overline{\text{err}} \equiv \sum_l \left(\frac{N_l}{N} \right) \sqrt{MSE_l}$$

where l indexes rectangles, N_l is the number of cases with $\mathbf{x}_i \in R_l$ and MSE_l is presumably from an OLS fit (of some linear model) in R_l .

2. Exactly what inputs x_j are used in each rectangle and how they are chosen is not clear. Output for an R implementation of Cubist lists different sets for the various rectangles.
3. Exactly how one goes from tree building to the final set of rules/rectangles is not clear. Software seems to not allow control of this. Perhaps there is some kind of combining of final rectangles from a tree.
4. Some sort of "smoothing" is involved. This seems to be some kind of averaging of regressions for bigger (containing) rectangles "up the tree branch" from a final rectangle. What this should mean is not absolutely clear if all one has is a set of "rules," particularly if there are cuts less

extreme than a final pair defining R_l that have been eliminated from description of R_l . For example

$$3 < x_1 < 5$$

is

$$3 < x_1 < 10 \text{ and } 3 < x_1 < 5$$

Further, the form of weights used in the averaging seems completely ad hoc.

There are two serious modifications of the basic "tree of regressions" notion that are included in the **R** implementation of Cubist:

1. One may employ "**committees**." This seems to be **boosting** or something much like it applied using the basic algorithm to create the corrections to successive versions of an approximate $E[y|\mathbf{x}]$.
2. One may employ "**instances**." This seems to be (optionally) applied after the boosting. It is shrinkage of \hat{y} s in light of y_i s for k -nearest neighbors, using weights depending upon the distances from \mathbf{x} to the neighbors. For k cases closest to \mathbf{x} , say cases i_1, i_2, \dots, i_k and corresponding weights w_1, w_2, \dots, w_k (summing to 1?) the prediction used for input \mathbf{x} seems from KJ to be³⁶

$$\hat{y}(\mathbf{x}) + \sum w_l (y_{i_l} - \hat{y}_{i_l})$$

A valuable general perspective that consideration of Quinlan's specific methods brings up might be called a **divide and conquer** strategy. In prediction problems where p is at all large, it is rare that one can find a simple form for a predictor that is effective across the entirety of an input space. One way to think about Quinlan's methods is as breaking an input space up into appropriate rectangles (defined by a tree structure) and then using primarily a (relatively simple) linear prediction form inside each rectangle. Of course, "the devil is in the details" of finding appropriate means of partitioning an input space and then simple forms to use in each piece of the space, but the general notion of solving several "local" prediction problems rather than a larger "global" one is clearly one that will on occasion be very effective. For, example, in a case where a few (say l) coordinates of an input vector \mathbf{x} are binary, it may make more sense to separately fit 2^l predictors (one for each possible binary vector) using the $p-l$ non-binary inputs instead of trying to fit a single predictor using the entire p -dimensional input.

³⁶This is a guess at a "correction" of a formula on KJ page 210 that seems incorrect.

Part III

Intermission: Perspective and Prediction in Practice

There is more to say about theory and specific methodology for statistical machine learning, but this is a sensible point at which to pause and reflect on the practice of prediction in "big data" contexts. Most of the best-known prediction methods have been discussed (the notable exception being linear classification methods and especially so-called support vector machines covered in the next chapter) and the basic concerns to be faced have been raised. The careful reader has what is needed in terms of statistical background to begin work on a large prediction problem. So here we provide a bit of summary discussion/perspective on beginning practice. (The material in the balance of these notes can be studied in parallel with practice on a large real problem. It is my belief that such practice grappling with the realities of prediction is essential to genuine understanding of modern statistical machine learning.)

The graphic in Figure 33 is intended to provide some conceptualization of what must be done to make predictions and honest judgments of how well they are likely to work. The graphic is meant to indicate that a project proceeds more or less left to right through it, but that actual practice is far too iterative and flexible to be adequately represented by a flowchart.

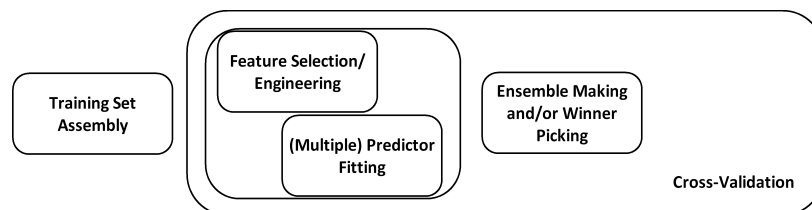


Figure 33: Elements of effective "big data" prediction

One must first assemble a training set from whatever sources are appropriate. Consistent with the "divide and conquer" discussion at the end of Section 11.5, this training set could represent only a well-defined part of a large input space and multiple graphics like Figure 33 in parallel would then in order. Note that if a breakup of the input space depends upon the data cases available (as in Quinlan's methodologies, where rectangles used depend upon the set of input vectors considered) that activity is best conceptualized as happening inside the big cross-validation box, perhaps before several parallel versions of what is presently Figure 33. The point is that the initial development of the training set is the conceptual base upon which all else is built and (at least if one is hoping to have reliable cross-validation results) a "random draws from a fixed universe" model must be a plausible description of both the elements of

the training set and additional "test" cases that are to be predicted.

Figure 33 puts "feature engineering" and "predictor fitting" activities inside a large single activity box. These are typically spoken of as if they were distinct, but they are largely indistinguishable/inseparable. (This is usually emphasized quite strongly by fans of neural network prediction, where the process of developing weights for linear combinations deep in the compositional structure of the predictor is often spoken of in terms of "learning good features" for prediction.)

The cross-validation box in Figure 33 encloses all but the assembling of the training set. This is a reminder of the basic principle that all that will ultimately be done to make a predictor must be done K times (on the K remainders) in order to create a reliable assessment of the likely effectiveness of a prediction methodology. Various "tuning" or "optimizing" steps based on some "cross-validation error" or "OOB error" measures may be employed in the fitting of a single one of multiple predictors in an ensemble, but only the kind of comprehensive "complete redoing" suggested by placing everything except training set assembly inside the largest activity box will be adequate as an indication of likely performance on new test cases.

Ultimately, producing good predictors in big real-world problems is a highly creative and interesting pursuit. What is presented in these notes amounts to a set of principles and building blocks that can be assembled in myriad ways. The fun is in finding clever problem-specific ways to do the assembly that prove to be practically effective.

Part IV

Supervised Learning II: More on Classification and Additional Theory

12 Basic Linear (and a Bit on Quadratic) Methods of Classification

Consider now methods of producing prediction/classification rules $\hat{f}(\mathbf{x})$ taking values in $\mathcal{G} = \{1, 2, \dots, K\}$ that have sets $\{\mathbf{x} \in \mathbb{R}^p \mid \hat{f}(\mathbf{x}) = k\}$ with boundaries that are (mostly) defined by linear equalities

$$\mathbf{x}'\boldsymbol{\beta} = c \tag{121}$$

The most obvious/naive potential method here is to regress K indicator variables $y_k = I[\hat{f}(\mathbf{x}) = k]$ onto \mathbf{x} (producing least squares regression vector coef-

ficients $\widehat{\boldsymbol{\beta}}_k$) and then to employ

$$\widehat{f}(\mathbf{x}) = \arg \max_k \widehat{f}_k(\mathbf{x}) = \arg \max_k \mathbf{x}' \widehat{\boldsymbol{\beta}}_k$$

But this often fails miserably because of the possibility of "masking" if $K > 2$. One must be smarter than this. Three kinds of smarter alternatives are Linear (and Quadratic) Discriminant Analysis, Logistic Regression, and direct searches for separating hyperplanes. The first two of these are "statistical" in origin with long histories in the field.

12.1 Linear (and a bit on Quadratic) Discriminant Analysis

Suppose that for $(\mathbf{x}, y) \sim P$, $\pi_k = P[y = k]$ and the conditional distribution of \mathbf{x} on \mathfrak{R}^p given that $y = k$ is $\text{MVN}_p(\boldsymbol{\mu}_k, \boldsymbol{\Sigma})$, i.e. the conditional pdf is

$$p(\mathbf{x}|k) = (2\pi)^{-p/2} (\det \boldsymbol{\Sigma})^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)' \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)\right)$$

Then it follows that

$$\ln\left(\frac{P[y = k|\mathbf{x}]}{P[y = l|\mathbf{x}]}\right) = \ln\left(\frac{\pi_k}{\pi_l}\right) - \frac{1}{2}\boldsymbol{\mu}'_k \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_k + \frac{1}{2}\boldsymbol{\mu}'_l \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_l + \mathbf{x}' \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_k - \boldsymbol{\mu}_l) \quad (122)$$

so that a theoretically optimal classifier/decision rule is

$$f(\mathbf{x}) = \arg \max_k \left[\ln(\pi_k) - \frac{1}{2}\boldsymbol{\mu}'_k \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_k + \mathbf{x}' \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_k \right]$$

and boundaries between regions in \mathfrak{R}^p where $f(\mathbf{x}) = k$ and $f(\mathbf{x}) = l$ are subsets of the sets

$$\left\{ \mathbf{x} \in \mathfrak{R}^p \mid \mathbf{x}' \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_k - \boldsymbol{\mu}_l) = -\ln\left(\frac{\pi_k}{\pi_l}\right) + \frac{1}{2}\boldsymbol{\mu}'_k \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}'_l \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_l \right\}$$

i.e. are defined by equalities of the form (121). Figure 34 illustrates this in a simple $K = 3$ and $p = 2$ context where all π_k s are the same.

This is dependent upon all K conditional normal distributions having the same covariance matrix, $\boldsymbol{\Sigma}$. In the event these are allowed to vary, conditional distribution k with covariance matrix $\boldsymbol{\Sigma}_k$, a theoretically optimal predictor/decision rule is

$$f(\mathbf{x}) = \arg \max_k \left[\ln(\pi_k) - \frac{1}{2} \ln(\det \boldsymbol{\Sigma}_k) - \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)' \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k) \right]$$

and boundaries between regions in \mathfrak{R}^p where $f(\mathbf{x}) = k$ and $f(\mathbf{x}) = l$ are subsets of the sets

$$\left\{ \mathbf{x} \in \mathfrak{R}^p \mid \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)' \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k) - \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_l)' \boldsymbol{\Sigma}_l^{-1}(\mathbf{x} - \boldsymbol{\mu}_l) = -\ln\left(\frac{\pi_k}{\pi_l}\right) - \frac{1}{2} \ln(\det \boldsymbol{\Sigma}_k) + \frac{1}{2} \ln(\det \boldsymbol{\Sigma}_l) \right\}$$

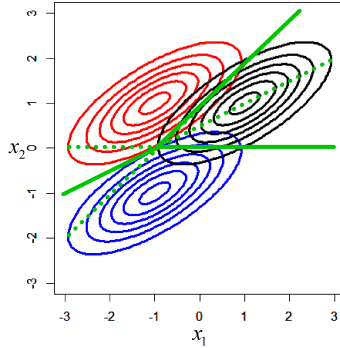


Figure 34: Contours of $K = 3$ bivariate normal pdfs and corresponding linear (equal class probability) classification boundaries.

Unless $\Sigma_k = \Sigma_l$ this kind of set is a quadratic surface in \mathcal{R}^p , not a hyperplane. One gets (not linear, but) Quadratic Discriminant Analysis.

Of course, in order to use LDA or QDA, one must estimate the vectors μ_k and the covariance matrix Σ or matrices Σ_k from the training data. Estimating K potentially different matrices Σ_k requires estimation of a very large number of parameters. So thinking about QDA versus LDA, one is again in the situation of needing to find the level of predictor complexity that a given dataset will support. QDA is a more flexible/complex method than LDA, but using it in preference to LDA increases the likelihood of overfit and poor prediction.

One idea that has been offered as a kind of continuous compromise between LDA and QDA is for $\alpha \in (0, 1)$ to use

$$\widehat{\Sigma}_k(\alpha) = \alpha \widehat{\Sigma}_k + (1 - \alpha) \widehat{\Sigma}_{\text{pooled}}$$

in place of $\widehat{\Sigma}_k$ in QDA. This kind of thinking even suggests as an estimate of a covariance matrix common across k

$$\widehat{\Sigma}(\gamma) = \gamma \widehat{\Sigma}_{\text{pooled}} + (1 - \gamma) \widehat{\sigma}^2 \mathbf{I}$$

for $\gamma \in (0, 1)$ and $\widehat{\sigma}^2$ an estimate of variance pooled across groups k and then across coordinates x_j of \mathbf{x} in LDA. Combining these two ideas, one might even invent a two-parameter set of fitted covariance matrices

$$\widehat{\Sigma}_k(\alpha, \gamma) = \alpha \widehat{\Sigma}_k + (1 - \alpha) \left(\gamma \widehat{\Sigma}_{\text{pooled}} + (1 - \gamma) \widehat{\sigma}^2 \mathbf{I} \right)$$

for use in QDA. Employing these in LDA or QDA provides the flexibility of choosing a complexity parameter or parameters and potentially improving prediction performance.

The form $\mathbf{x}'\boldsymbol{\beta}$ is (of course and by design) linear in the coordinates of \mathbf{x} . An obvious natural generalization of this discussion is to consider discriminants that

are linear in some (non-linear) functions of the coordinates of \mathbf{x} . This is simply choosing some M basis functions/transforms/features $h_m(\mathbf{x})$ and replacing the p coordinates of \mathbf{x} with the M coordinates of $(h_1(\mathbf{x}), h_2(\mathbf{x}), \dots, h_M(\mathbf{x}))$ in the development of LDA.

Of course, upon choosing basis functions that are all coordinates, squares of coordinates, and products of coordinates of \mathbf{x} , one produces *linear* (in the basis functions) discriminants that are general *quadratic* functions of \mathbf{x} . The possibilities opened here are myriad and (as always) "the devil is in the details."

12.1.1 Dimension Reduction in LDA

Where p is large, a common methodology in LDA is forward selection of coordinates x_j of \mathbf{x} to use in classification. Cross-validation can be used to choose a best number of coordinates and potentially achieve some dimension-reduction and reduce overfitting.

Another idea in the direction of simplifying the interpretation of LDA by dimension-reduction is use of "canonical coordinates" and intends to replace "variable selection" with use of a hopefully few relevant linear combinations of coordinates (producing "reduced rank LDA"). Let

$$\bar{\boldsymbol{\mu}} = \frac{1}{K} \sum_{k=1}^K \boldsymbol{\mu}_k$$

and note that one is free to replace \mathbf{x} and all K means $\boldsymbol{\mu}_k$ with respectively

$$\mathbf{x}^* = \boldsymbol{\Sigma}^{-1/2} (\mathbf{x} - \bar{\boldsymbol{\mu}}) \quad \text{and} \quad \boldsymbol{\mu}_k^* = \boldsymbol{\Sigma}^{-1/2} (\boldsymbol{\mu}_k - \bar{\boldsymbol{\mu}})$$

This produces

$$\ln \left(\frac{P[y = k | \mathbf{x}^*]}{P[y = l | \mathbf{x}^*]} \right) = \ln \left(\frac{\pi_k}{\pi_l} \right) - \frac{1}{2} \|\mathbf{x}^* - \boldsymbol{\mu}_k^*\|^2 + \frac{1}{2} \|\mathbf{x}^* - \boldsymbol{\mu}_l^*\|^2$$

and (in "sphered" form) the theoretically optimal classifier can be described as

$$f(\mathbf{x}) = \arg \max_k \left[\ln(\pi_k) - \frac{1}{2} \|\mathbf{x}^* - \boldsymbol{\mu}_k^*\|^2 \right]$$

That is, in terms of \mathbf{x}^* , optimal decisions are based on ordinary Euclidian distances to the transformed means $\boldsymbol{\mu}_k^*$. Further, this form can often be made even simpler/be seen to depend upon a lower-dimensional (than p) distance.

The $\boldsymbol{\mu}_k^*$ typically span a subspace of \Re^p of dimension $\min(p, K - 1)$. For

$$\mathbf{M}_{p \times K} = (\boldsymbol{\mu}_1^*, \boldsymbol{\mu}_2^*, \dots, \boldsymbol{\mu}_K^*)$$

let \mathbf{P}_M be the $p \times p$ projection matrix projecting onto the column space of \mathbf{M} in \Re^p ($C(\mathbf{M})$). Then

$$\begin{aligned} \|\mathbf{x}^* - \boldsymbol{\mu}_k^*\|^2 &= \|[\mathbf{P}_M + (\mathbf{I} - \mathbf{P}_M)](\mathbf{x}^* - \boldsymbol{\mu}_k^*)\|^2 \\ &= \|(\mathbf{P}_M \mathbf{x}^* - \boldsymbol{\mu}_k^*) + (\mathbf{I} - \mathbf{P}_M) \mathbf{x}^*\|^2 \\ &= \|\mathbf{P}_M \mathbf{x}^* - \boldsymbol{\mu}_k^*\|^2 + \|(\mathbf{I} - \mathbf{P}_M) \mathbf{x}^*\|^2 \end{aligned}$$

the last equality coming because $(\mathbf{P}_M \mathbf{x}^* - \boldsymbol{\mu}_k^*) \in C(\mathbf{M})$ and $(\mathbf{I} - \mathbf{P}_M) \mathbf{x}^* \in C(\mathbf{M})^\perp$. Since $\|(\mathbf{I} - \mathbf{P}_M) \mathbf{x}^*\|^2$ doesn't depend upon k , the theoretically optimal predictor/decision rule can be described as

$$f(\mathbf{x}) = \arg \max_k \left[\ln(\pi_k) - \frac{1}{2} \|\mathbf{P}_M \mathbf{x}^* - \boldsymbol{\mu}_k^*\|^2 \right]$$

and theoretically optimal decision rules can be described in terms of the projection of \mathbf{x}^* onto $C(\mathbf{M})$ and its distances to the $\boldsymbol{\mu}_k^*$.

Now,

$$\frac{1}{K} \mathbf{M} \mathbf{M}'$$

is the (typically rank $\min(p, K - 1)$) sample covariance matrix of the $\boldsymbol{\mu}_k^*$ and has an eigen decomposition as

$$\frac{1}{K} \mathbf{M} \mathbf{M}' = \mathbf{V} \mathbf{D} \mathbf{V}'$$

for

$$\mathbf{D} = \mathit{diag}(d_1, d_2, \dots, d_p)$$

where

$$d_1 \geq d_2 \geq \dots \geq d_p$$

are the eigenvalues and the columns of \mathbf{V} are orthonormal eigenvectors corresponding in order to the successively smaller eigenvalues of $\frac{1}{K} \mathbf{M} \mathbf{M}'$. These \mathbf{v}_k with $d_k > 0$ specify linear combinations of the coordinates of the $\boldsymbol{\mu}_l^*$, $\langle \mathbf{v}_k, \boldsymbol{\mu}_l^* \rangle$, with the largest possible sample variances subject to the constraints that $\|\mathbf{v}\| = 1$ and $\langle \mathbf{v}_l, \mathbf{v}_k \rangle = 0$ for all $l < k$. These \mathbf{v}_k are perpendicular vectors in successive directions of most important unaccounted-for spread of the $\boldsymbol{\mu}_k^*$.

Then, for $l \leq \text{rank}(\mathbf{M} \mathbf{M}')$ define

$$\mathbf{V}_l = (\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_l)$$

let

$$\mathbf{P}_l = \mathbf{V}_l \mathbf{V}_l'$$

be the matrix projecting onto $C(\mathbf{V}_l)$ in \Re^p . A possible "reduced rank" approximation to the theoretically optimal LDA classification rule is

$$f_l(\mathbf{x}) = \arg \max_k \left[\ln(\pi_k) - \frac{1}{2} \|\mathbf{P}_l \mathbf{x}^* - \mathbf{P}_l \boldsymbol{\mu}_k^*\|^2 \right]$$

and l becomes a complexity parameter that one might optimize via cross-validation to tune or regularize the method.

Note also that for $\mathbf{w} \in \Re^p$

$$\mathbf{P}_l \mathbf{w} = \sum_{k=1}^l \langle \mathbf{v}_k, \mathbf{w} \rangle \mathbf{v}_k$$

For purposes of graphical representation of what is going on in these computations, one might replace the p coordinates of \mathbf{x} and the means $\boldsymbol{\mu}_k$ with the l coordinates of

$$(\langle \mathbf{v}_1, \mathbf{x}^* \rangle, \langle \mathbf{v}_2, \mathbf{x}^* \rangle, \dots, \langle \mathbf{v}_l, \mathbf{x}^* \rangle)' \quad (123)$$

and of the

$$(\langle \mathbf{v}_1, \boldsymbol{\mu}_k^* \rangle, \langle \mathbf{v}_2, \boldsymbol{\mu}_k^* \rangle, \dots, \langle \mathbf{v}_l, \boldsymbol{\mu}_k^* \rangle)' \quad (124)$$

(that might be called "canonical coordinates"). It seems to be ordered pairs of entries of these vectors that are plotted by HTF in their Figures 4.8 and 4.11. In this regard, we need to point out that since any eigenvector \mathbf{v}_k could be replaced by $-\mathbf{v}_k$ without any fundamental effect in the above development, the vector (123) and all of the vectors (124) could be altered by multiplication of any particular set of coordinates by -1 . (Whether a particular algorithm for finding eigenvectors produces \mathbf{v}_k or $-\mathbf{v}_k$ is not fundamental, and there seems to be no standard convention in this regard.) It appears that the pictures in HTF might have been made using the R function `lda` and its choice of signs for eigenvectors.

12.2 Logistic Regression

A generalization of the MVN conditional distribution result (122) is an *assumption* that for all $k < K$

$$\ln \left(\frac{P[y = k|\mathbf{x}]}{P[y = K|\mathbf{x}]} \right) = \beta_{k0} + \mathbf{x}'\boldsymbol{\beta}_k \quad (125)$$

Here there are $K - 1$ constants β_{k0} and $K - 1$ p -vectors $\boldsymbol{\beta}_k$ to be specified, not necessarily tied to class mean vectors or a common within-class covariance matrix for \mathbf{x} . In fact, the set of relationships (125) do not fully specify a joint distribution for (\mathbf{x}, y) . Rather, they only specify the nature of the conditional distributions of $y|\mathbf{x}$. (In this regard, the situation is exactly analogous to that in ordinary simple linear regression. A bivariate normal distribution for (x, y) gets one normal conditional distributions for y with a constant variance and mean linear in x . But one may make those assumptions conditionally on x , without assuming anything about the marginal distribution of x , that in the bivariate normal model is univariate normal.)

Using $\boldsymbol{\theta}$ as shorthand for a vector containing all the constants β_{k0} and the vectors $\boldsymbol{\beta}_k$, the linear log probability ratio assumption (125) produces the forms

$$P[y = k|\mathbf{x}] = p_k(\mathbf{x}, \boldsymbol{\theta}) = \frac{\exp(\beta_{k0} + \mathbf{x}'\boldsymbol{\beta}_k)}{1 + \sum_{k=1}^{K-1} \exp(\beta_{k0} + \mathbf{x}'\boldsymbol{\beta}_k)} \quad (126)$$

for $k < K$, and

$$P[y = K|\mathbf{x}] = p_K(\mathbf{x}, \boldsymbol{\theta}) = \frac{1}{1 + \sum_{k=1}^{K-1} \exp(\beta_{k0} + \mathbf{x}'\boldsymbol{\beta}_k)} \quad (127)$$

and a theoretically optimal (under 0-1 loss) predictor/classification rule is

$$f(\mathbf{x}) = \arg \max_k p_k(\mathbf{x}, \boldsymbol{\theta})$$

As a bit of an aside, it is perhaps useful to see in forms (126) and (127) use of the softmax function with linear combinations of the coordinates of \mathbf{x} and be reminded of the neural network discussion of Section 8.2. In that regard, consider an extremely simple neural network for classification having no hidden layers and all coefficients for the last output node set to 0. That is, with no hidden layers, if in the notation of Section 8.3.1 the last column of \mathbf{A}^0 by assumption contains only 0s ($\mathbf{A}_K^0 = \mathbf{0}$), the corresponding "neural network" for classification is exactly the K -class logistic regression model.

Figure 35 is a plot of three different $p = 1$ forms for $p_1(x, \beta_0, \beta_1)$ in a $K = 2$ model. The parameter sets are

Red: $\beta_0 = 0, \beta_1 = 1,$
 Blue: $\beta_0 = -4, \beta_1 = 2,$ and
 Green: $\beta_0 = -2, \beta_1 = -2$

In each case $p_1(x, \beta_0, \beta_1) = .5$ where $x = -\beta_0/\beta_1$, the function increases in x exactly when $\beta_1 > 0$, and curve steepness increases with $|\beta_1|$.

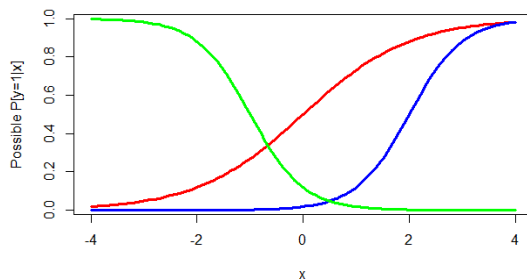


Figure 35: Plot of three different $p = 1$ forms for $p_1(x, \beta_0, \beta_1)$ in a $K = 2$ model.

In a $K = 2$ case with $p = 2$, (for its $\{1, 2\}$ coding of y) the kind of relationship pictured in Figure 36 holds. $p_1(\mathbf{x}, \beta_0, \beta_1, \beta_2)$ defines an "s-shaped surface" that is "steep" when coefficients β_1, β_2 have large absolute values, is constant on lines $\beta_0 + \beta_1 x_1 + \beta_2 x_2 = c$ in \mathfrak{R}^2 , taking the value .5 on the line $\beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0$.

Assumption (125) generalizes the "mixture of MVNs" assumption of LDA, and standard methods of fitting the corresponding parameters based on training data are necessarily fundamentally different. That is (using maximum likelihood) in LDA, the K probabilities π_k , the K means $\boldsymbol{\mu}_k$, and the covariance matrix $\boldsymbol{\Sigma}$ might be chosen to maximize the likelihood

$$\prod_{i=1}^N \pi_{y_i} p(\mathbf{x}_i | \boldsymbol{\mu}_{y_i}, \boldsymbol{\Sigma})$$

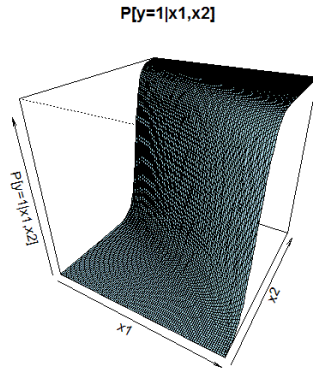


Figure 36: A plot of a $p = 2$ form for $p_1(x, \beta_0, \beta_1)$ in a $K = 2$ model (with 1-2 coding).

This is a mixture model and the complete likelihood is involved, i.e. a joint density for the N pairs (\mathbf{x}_i, y_i) . On the other hand, standard logistic regression methodology maximizes

$$\prod_{i=1}^N p_{y_i}(\mathbf{x}_i, \boldsymbol{\theta}) \quad (128)$$

over choices of $\boldsymbol{\theta}$. This is not a full likelihood, but rather one *conditional* on the \mathbf{x}_i observed.

In a $K = 2$ case with $-1-1$ coding for y , the logistic regression log-likelihood has a very simple form. With

$$p_{-1}(\mathbf{x}, \beta_0, \boldsymbol{\beta}) = \frac{\exp(\beta_0 + \mathbf{x}'\boldsymbol{\beta})}{1 + \exp(\beta_0 + \mathbf{x}'\boldsymbol{\beta})} \quad \text{and} \quad p_1(\mathbf{x}, \beta_0, \boldsymbol{\beta}) = \frac{1}{1 + \exp(\beta_0 + \mathbf{x}'\boldsymbol{\beta})}$$

the likelihood term contributed to the product (128) by (\mathbf{x}_i, y_i) is

$$\left(\frac{\exp(\beta_0 + \mathbf{x}'_i \boldsymbol{\beta})}{1 + \exp(\beta_0 + \mathbf{x}'_i \boldsymbol{\beta})} \right)^{I[y_i = -1]} \left(\frac{1}{1 + \exp(\beta_0 + \mathbf{x}'_i \boldsymbol{\beta})} \right)^{I[y_i = 1]}$$

It then follows that the contribution of (\mathbf{x}_i, y_i) to the log-likelihood is

$$\begin{aligned} & I[y_i = -1] (\beta_0 + \mathbf{x}'_i \boldsymbol{\beta}) - \ln(1 + \exp(\beta_0 + \mathbf{x}'_i \boldsymbol{\beta})) \\ &= -\ln(1 + \exp(y_i (\beta_0 + \mathbf{x}'_i \boldsymbol{\beta}))) \end{aligned}$$

(Note that this term is $-(\ln 2) h_1(y_i (\beta_0 + \mathbf{x}'_i \boldsymbol{\beta}))$ for h_1 the first of the function "losses" considered in Section 1.5.3 in the discussion of voting functions in 2-class classification.) So ultimately, the $K = 2$ log-likelihood (to be optimized in ML fitting) is

$$-\sum_{i=1}^N \ln(1 + \exp(y_i (\beta_0 + \mathbf{x}'_i \boldsymbol{\beta})))$$

(which is $-\ln 2$ times the total loss in the gradient boosting algorithm applied to voting function $g(\mathbf{x}) = \beta_0 + \mathbf{x}'\boldsymbol{\beta}$).

A general alternative to maximum likelihood (useful in avoiding overfitting for large N) is *minimization* of a criterion like

$$-\ln \left(\prod_{i=1}^N p_{y_i}(\mathbf{x}_i, \boldsymbol{\theta}) \right) + \text{penalty}(\boldsymbol{\theta})$$

For example, in the $K = 2$ case (with $-1-1$ coding) a lasso version is (for $\lambda > 0$ and $0 \leq \alpha \leq 1$) minimization of

$$\sum_{i=1}^N \ln(1 + \exp(y_i(\beta_0 + \mathbf{x}'_i\boldsymbol{\beta}))) + \lambda \left(\alpha \sum_{j=1}^p |\beta_j| + \frac{(1-\alpha)}{2} \sum_{j=1}^p \beta_j^2 \right)$$

(that can be accomplished in **R** using `glmnet`).

It is common to encounter situations where (say in a $K = 2$ context with 0-1 coding) π_0 is quite small. Rather than trying to do analysis on a random sample of (\mathbf{x}, y) pairs where there would be relatively few $y = 0$ cases, there are a number of potentially important practical reasons for doing analysis of a dataset consisting of random sample of N_0 instances ("cases") with $y = 0$ and a random sample of N_1 instances ("controls") with $y = 1$, where $N_0/(N_0 + N_1)$ is nowhere nearly as small as π_0 .³⁷ (In fact, N_1 on the order of 5 or 6 times N_0 is often recommended.)

For $K = 2$

$$\ln \left(\frac{P[y = 0|\mathbf{x}]}{P[y = 1|\mathbf{x}]} \right) = \ln \left(\frac{\pi_0 p(\mathbf{x}|0)}{\pi_1 p(\mathbf{x}|1)} \right) = \ln \left(\frac{\pi_0}{\pi_1} \right) + \ln \left(\frac{p(\mathbf{x}|0)}{p(\mathbf{x}|1)} \right)$$

So under the logistic regression assumption that

$$\ln \left(\frac{P[y = 0|\mathbf{x}]}{P[y = 1|\mathbf{x}]} \right) = \beta_0 + \mathbf{x}'\boldsymbol{\beta}$$

fitting to a case-control dataset should produce

$$\begin{aligned} \hat{\beta}_0^{\text{cc}} + \mathbf{x}'\hat{\boldsymbol{\beta}}^{\text{cc}} &\approx \ln \left(\frac{N_0}{N_1} \right) + \ln \left(\frac{p(\mathbf{x}|0)}{p(\mathbf{x}|1)} \right) \\ &= \ln \left(\frac{P[y = 0|\mathbf{x}]}{P[y = 1|\mathbf{x}]} \right) + \ln \left(\frac{N_0}{N_1} \right) - \ln \left(\frac{\pi_0}{\pi_1} \right) \end{aligned}$$

So (presuming that an estimate $\hat{\pi}_0$ is available) estimated coefficients

$$\hat{\beta}_0 \equiv \hat{\beta}_0^{\text{cc}} - \ln \left(\frac{N_0}{N_1} \right) + \ln \left(\frac{\hat{\pi}_0}{1 - \hat{\pi}_0} \right) \quad \text{and} \quad \hat{\boldsymbol{\beta}} = \hat{\boldsymbol{\beta}}^{\text{cc}}$$

³⁷Notice that this methodology purposely creates a situation like that described in Section 1.5.1, where training set class relative frequencies are much different from actual class probabilities.

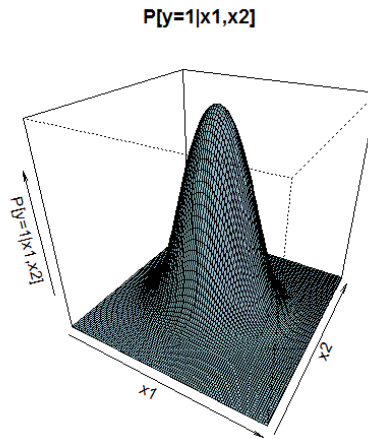


Figure 37: An example of a quadratic form $(-.2x_1^2 - .3x_2^2)$ used to make logistic regression probabilities that $y = 1$ (for 1-2 coding)

are appropriate for the original context. (This result is a specialization of the general formula (29) for shifting conditional probabilities for $y|\mathbf{x}$ based on use of a training set with class frequencies different from the π_k s.)

Good logistic regression models are the basis of good classifiers when one classifies according to the largest predicted probability. And just as the usefulness of LDA can be extended by consideration of transforms/features made from an original p -dimensional \mathbf{x} , the same is true for logistic regression. For example, beginning with x_1 and x_2 and creating additional predictors x_1^2, x_2^2 , and x_1x_2 , one can use logistic regression technology based on the 5-dimensional input $(x_1, x_2, x_1^2, x_2^2, x_1x_2)$ to create classification boundaries that are **quadratic** in terms of the original x_1 and x_2 . An example of the kind of functional form for the conditional probability that $y = k$ given a bivariate input \mathbf{x} that can result is portrayed in Figure 37 where the quadratic form $-.2x_1^2 - .3x_2^2$ is used to make logistic regression probabilities that $y = 1$ (for 1-2 coding). Constant-probability contours of such a surface are ellipses in (x_1, x_2) -space.

12.3 Separating Hyperplanes

In the $K = 2$ group case now use the $\mathcal{G} = \{-1, 1\}$ coding. If there is a $\beta \in \mathfrak{R}^p$ and real number β_0 such that in the training data

$$y = 1 \text{ exactly when } \mathbf{x}'\beta + \beta_0 > 0$$

a "separating hyperplane"

$$\{\mathbf{x} \in \mathfrak{R}^p \mid \mathbf{x}'\beta + \beta_0 = 0\}$$

can be found via logistic regression. The (conditional) likelihood will not have a maximum, but if one follows a search path far enough toward the limiting

value of 0 for the loglikelihood or 1 for the likelihood, satisfactory $\beta \in \mathfrak{R}^p$ and β_0 from an iteration of the search algorithm will produce separation.

A famous older algorithm for finding a separating hyperplane is the so-called "perceptron" algorithm. It can be defined as follows. From some starting points β^0 and β_0^0 cycle through the training data cases in order (repeatedly as needed). At any iteration l , take

$$\left\{ \begin{array}{l} \beta^l = \beta^{l-1} \text{ and } \beta_0^l = \beta_0^{l-1} \\ \beta^l = \beta^{l-1} + y_i \mathbf{x}_i \\ \text{and } \beta_0^l = \beta_0^{l-1} + y_i \end{array} \right\} \quad \text{if } \left\{ \begin{array}{l} y_i = 1 \text{ and } \mathbf{x}'_i \beta + \beta_0 > 0, \text{ or} \\ y_i = -1 \text{ and } \mathbf{x}'_i \beta + \beta_0 \leq 0 \end{array} \right\}$$

$$\left. \begin{array}{l} \\ \\ \end{array} \right\} \quad \text{otherwise}$$

This will eventually identify a separating hyperplane when a series of N iterations fails to change the values of β and β_0 .

If there is a separating hyperplane, it will typically not be unique. One can attempt to define and search for "optimal" such hyperplanes that, e.g., maximize distance from the plane to the closest training vector. The material on "support vector classifiers" in Section 13.1 is a famous development in this direction.

13 Support Vector Machines

Consider a 2-class classification problem. For notational convenience, we'll suppose that output y takes values in $\mathcal{G} = \{-1, 1\}$. Our present concern is in a further development of linear classification methodology beyond that provided in Section 12.

For $\beta \in \mathfrak{R}^p$ and $\beta_0 \in \mathfrak{R}$ we'll consider the voting function

$$g(\mathbf{x}) = \mathbf{x}'\beta + \beta_0 \tag{129}$$

and a theoretical predictor/classifier

$$f(\mathbf{x}) = \text{sign}(g(\mathbf{x})) \tag{130}$$

We will approach the problem of choosing β and β_0 to in some sense provide a maximal cushion around a hyperplane separating between \mathbf{x}_i with corresponding $y_i = -1$ and \mathbf{x}_i with corresponding $y_i = 1$.

13.1 The Linearly Separable Case: Maximum Margin Classifiers

In the case that there is a classifier of form (130) with 0 training error rate, we consider the optimization problem

$$\begin{array}{ll} \text{maximize} & M \\ \text{with } \|\mathbf{u}\| = 1 & \text{subject to } y_i(\mathbf{x}'_i \mathbf{u} + \beta_0) \geq M \quad \forall i \\ \text{and } \beta_0 \in \mathfrak{R} & \end{array} \tag{131}$$

This can be thought of in terms of choosing a unit vector \mathbf{u} (or direction) in \mathfrak{R}^p so that upon projecting the training input vectors \mathbf{x}_i onto the subspace of multiples of \mathbf{u} there is maximum separation between the convex hull of projections of the \mathbf{x}_i with $y_i = -1$ and the convex hull of projections of \mathbf{x}_i with corresponding $y_i = 1$. (The sign on \mathbf{u} is chosen to give the latter larger $\mathbf{x}'_i \mathbf{u}$ than the former.) If \mathbf{u} and β_0 solve this maximization problem the (maximum) **margin** is then

$$M = \frac{1}{2} \left(\min_{\substack{\mathbf{x}_i \text{ with} \\ y_i = 1}} \mathbf{x}'_i \mathbf{u} - \max_{\substack{\mathbf{x}_i \text{ with} \\ y_i = -1}} \mathbf{x}'_i \mathbf{u} \right)$$

and the constant that makes the voting function (129) take the value 0 is

$$\beta_0 = -\frac{1}{2} \left(\min_{\substack{\mathbf{x}_i \text{ with} \\ y_i = 1}} \mathbf{x}'_i \mathbf{u} + \max_{\substack{\mathbf{x}_i \text{ with} \\ y_i = -1}} \mathbf{x}'_i \mathbf{u} \right)$$

The geometry of this formalism in a small $p = 2$ case is illustrated in Figure 38.

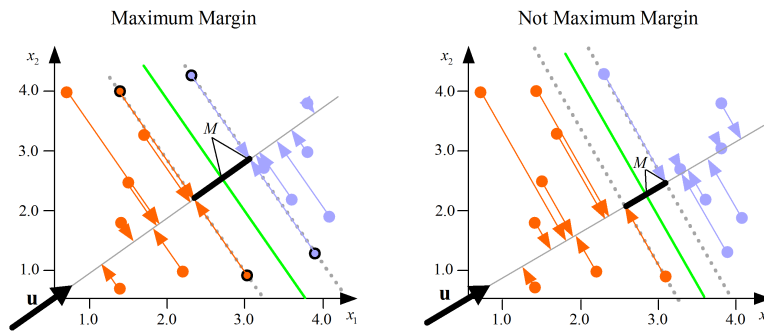


Figure 38: The geometry of maximum margin classification for a small $p = 2$ example.

For purposes of applying standard optimization theory and software, it is useful to reformulate the basic problem (131) several ways. First, note that optimization problem (131) may be rewritten as

$$\begin{aligned} & \text{maximize} && M & \text{subject to} && y_i \left(\mathbf{x}'_i \left(\frac{\mathbf{u}}{M} \right) + \frac{\beta_0}{M} \right) \geq 1 \quad \forall i && (132) \\ & \text{with} && \|\mathbf{u}\| = 1 && && && \\ & \text{and} && \beta_0 \in \mathfrak{R} && && && \end{aligned}$$

Then if we let

$$\boldsymbol{\beta} = \frac{\mathbf{u}}{M}$$

it's the case that

$$\|\boldsymbol{\beta}\| = \frac{1}{M} \quad \text{or} \quad M = \frac{1}{\|\boldsymbol{\beta}\|}$$

so that problem (132) can be rewritten

$$\begin{aligned} & \underset{\boldsymbol{\beta} \in \Re^p}{\text{minimize}} \quad \frac{1}{2} \|\boldsymbol{\beta}\|^2 \quad \text{subject to } y_i (\mathbf{x}'_i \boldsymbol{\beta} + \beta_0) \geq 1 \quad \forall i \quad (133) \\ & \text{and } \beta_0 \in \Re \end{aligned}$$

This formulation (133) is that of a convex (quadratic criterion, linear inequality constraints) optimization problem for which there exists standard theory and algorithms.

The so-called primal functional corresponding to problem (133) is (for $\boldsymbol{\alpha} \in \Re^N$)

$$F_P(\boldsymbol{\beta}, \beta_0, \boldsymbol{\alpha}) \equiv \frac{1}{2} \|\boldsymbol{\beta}\|^2 - \sum_{i=1}^N \alpha_i (y_i (\mathbf{x}'_i \boldsymbol{\beta} + \beta_0) - 1) \quad \text{for } \boldsymbol{\alpha} \geq \mathbf{0}$$

To solve problem (133), one may for each $\boldsymbol{\alpha} \geq \mathbf{0}$ choose $(\boldsymbol{\beta}(\boldsymbol{\alpha}), \beta_0(\boldsymbol{\alpha}))$ to minimize $F_P(\cdot, \cdot, \boldsymbol{\alpha})$ and then choose $\boldsymbol{\alpha} \geq \mathbf{0}$ to *maximize* $F_P(\boldsymbol{\beta}(\boldsymbol{\alpha}), \beta_0(\boldsymbol{\alpha}), \boldsymbol{\alpha})$. The Karush-Kuhn-Tucker conditions are necessary and sufficient for solution of a constrained optimization problem. In the present context they are the *gradient conditions*

$$\frac{\partial F_P(\boldsymbol{\beta}, \beta_0, \boldsymbol{\alpha})}{\partial \beta_0} = - \sum_{i=1}^N \alpha_i y_i = 0 \quad (134)$$

and

$$\frac{\partial F_P(\boldsymbol{\beta}, \beta_0, \boldsymbol{\alpha})}{\partial \boldsymbol{\beta}} = \boldsymbol{\beta} - \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i = \mathbf{0} \quad (135)$$

the *feasibility conditions*

$$y_i (\mathbf{x}'_i \boldsymbol{\beta} + \beta_0) - 1 \geq 0 \quad \forall i \quad (136)$$

the *non-negativity conditions*

$$\boldsymbol{\alpha} \geq \mathbf{0} \quad (137)$$

and the *orthogonality conditions*

$$\alpha_i (y_i (\mathbf{x}'_i \boldsymbol{\beta} + \beta_0) - 1) = 0 \quad \forall i \quad (138)$$

Now relationships (134) and (135) are respectively

$$\sum_{i=1}^N \alpha_i y_i = 0 \quad \text{and} \quad \boldsymbol{\beta} = \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i \equiv \boldsymbol{\beta}(\boldsymbol{\alpha}) \quad (139)$$

and plugging these into $F_P(\boldsymbol{\beta}, \beta_0, \boldsymbol{\alpha})$ gives a function of $\boldsymbol{\alpha}$ only

$$\begin{aligned}
F_D(\boldsymbol{\alpha}) &\equiv \frac{1}{2} \|\boldsymbol{\beta}(\boldsymbol{\alpha})\|^2 - \sum_{i=1}^N \alpha_i (y_i \mathbf{x}'_i \boldsymbol{\beta}(\boldsymbol{\alpha}) - 1) \\
&= \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}'_i \mathbf{x}_j - \sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}'_i \mathbf{x}_j + \sum_i \alpha_i \\
&= \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}'_i \mathbf{x}_j \\
&= \mathbf{1}' \boldsymbol{\alpha} - \frac{1}{2} \boldsymbol{\alpha}' \mathbf{H} \boldsymbol{\alpha}
\end{aligned}$$

for

$$\mathbf{H}_{N \times N} = (y_i y_j \mathbf{x}'_i \mathbf{x}_j) \quad (140)$$

Then the "dual" problem for problem (133) is the N -dimensional optimization problem

$$\text{maximize}_{\boldsymbol{\alpha} \in \mathbb{R}^N} \mathbf{1}' \boldsymbol{\alpha} - \frac{1}{2} \boldsymbol{\alpha}' \mathbf{H} \boldsymbol{\alpha} \quad \text{subject to } \boldsymbol{\alpha} \geq \mathbf{0} \text{ and } \boldsymbol{\alpha}' \mathbf{y} = 0 \quad (141)$$

and apparently this problem is easily solved.

Now condition (138) implies that if $\alpha_i^{\text{opt}} > 0$

$$y_i (\mathbf{x}'_i \boldsymbol{\beta}(\boldsymbol{\alpha}^{\text{opt}}) + \beta_0(\boldsymbol{\alpha}^{\text{opt}})) = 1$$

so that

1. by condition (136) the corresponding \mathbf{x}_i has minimum $\mathbf{x}'_i \boldsymbol{\beta}(\boldsymbol{\alpha}^{\text{opt}})$ for training vectors with $y_i = 1$ or maximum $\mathbf{x}'_i \boldsymbol{\beta}(\boldsymbol{\alpha}^{\text{opt}})$ for training vectors with $y_i = -1$ (so that \mathbf{x}_i is a **support vector** for the "slab" of thickness $2M$ around a separating hyperplane),
2. $\beta_0(\boldsymbol{\alpha}^{\text{opt}})$ may be determined using the corresponding \mathbf{x}_i from

$$y_i \beta_0(\boldsymbol{\alpha}^{\text{opt}}) = 1 - y_i \mathbf{x}'_i \boldsymbol{\beta}(\boldsymbol{\alpha}^{\text{opt}}) \quad \text{i.e.} \quad \beta_0(\boldsymbol{\alpha}^{\text{opt}}) = y_i - \mathbf{x}'_i \boldsymbol{\beta}(\boldsymbol{\alpha}^{\text{opt}})$$

(apparently for reasons of numerical stability it is common practice to average values $y_i - \mathbf{x}'_i \boldsymbol{\beta}(\boldsymbol{\alpha}^{\text{opt}})$ for support vectors in order to evaluate $\beta_0(\boldsymbol{\alpha}^{\text{opt}})$), and

- 3.

$$\begin{aligned}
1 &= y_i \beta_0(\boldsymbol{\alpha}^{\text{opt}}) + y_i \left(\sum_{j=1}^N \alpha_j^{\text{opt}} y_j \mathbf{x}_j \right)' \mathbf{x}_i \\
&= y_i \beta_0(\boldsymbol{\alpha}^{\text{opt}}) + \sum_{j=1}^N \alpha_j^{\text{opt}} y_j y_i \mathbf{x}'_j \mathbf{x}_i
\end{aligned}$$

The fact (139) that $\boldsymbol{\beta}(\boldsymbol{\alpha}) \equiv \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i$ implies that only the training cases with $\alpha_i > 0$ (typically corresponding to a relatively few support vectors) determine the nature of the solution to this optimization problem. Further, for \mathcal{SV} the set indices of support vectors in the problem,

$$\begin{aligned} \|\boldsymbol{\beta}(\boldsymbol{\alpha}^{\text{opt}})\|^2 &= \sum_{i \in \mathcal{SV}} \sum_{j \in \mathcal{SV}} \alpha_i^{\text{opt}} \alpha_j^{\text{opt}} y_i y_j \mathbf{x}'_i \mathbf{x}_j \\ &= \sum_{i \in \mathcal{SV}} \alpha_i^{\text{opt}} \sum_{j \in \mathcal{SV}} \alpha_j^{\text{opt}} y_i y_j \mathbf{x}'_i \mathbf{x}_j \\ &= \sum_{i \in \mathcal{SV}} \alpha_i^{\text{opt}} (1 - y_i \beta_0(\boldsymbol{\alpha}^{\text{opt}})) \\ &= \sum_{i \in \mathcal{SV}} \alpha_i^{\text{opt}} \end{aligned}$$

the next to last of these equalities following from 3. above, and the last following from the gradient condition (134). Then the margin for this problem is simply

$$M = \frac{1}{\|\boldsymbol{\beta}(\boldsymbol{\alpha}^{\text{opt}})\|} = \frac{1}{\sqrt{\sum_{i \in \mathcal{SV}} \alpha_i^{\text{opt}}}} \quad (142)$$

13.2 The Linearly Non-separable Case: Support Vector Classifiers

In a linearly non-separable case, the convex optimization problem (133) does not have a solution (no pair $\boldsymbol{\beta} \in \mathfrak{R}^p$ and $\beta_0 \in \mathfrak{R}$ provides $y_i (\mathbf{x}'_i \boldsymbol{\beta} + \beta_0) \geq 1 \ \forall i$). We might, therefore (in looking for good choices of $\boldsymbol{\beta} \in \mathfrak{R}^p$ and $\beta_0 \in \mathfrak{R}$) try to relax the constraints of the problem slightly. That is, suppose that $\xi_i \geq 0$ for $i = 1, 2, \dots, N$ and consider the set of constraints

$$y_i (\mathbf{x}'_i \boldsymbol{\beta} + \beta_0) + \xi_i \geq 1 \ \forall i$$

(the ξ_i are called "slack" variables and provide some "wobble room" in search for a hyperplane that "nearly" separates the two classes with a good margin). We might try to control the total amount of slack allowed by setting a bound

$$\sum_{i=1}^N \xi_i \leq C$$

for some positive C (a "budget").

Note that if $y_i (\mathbf{x}'_i \boldsymbol{\beta} + \beta_0) \geq 0$, case i is correctly classified in the training set, and so if for some pair $\boldsymbol{\beta} \in \mathfrak{R}^p$ and $\beta_0 \in \mathfrak{R}$ this holds for all i , we have a separable problem. So any non-separable problem must have at least one negative $y_i (\mathbf{x}'_i \boldsymbol{\beta} + \beta_0)$ for any $\boldsymbol{\beta} \in \mathfrak{R}^p$ and $\beta_0 \in \mathfrak{R}$ pair. This in turn requires that the budget C must be at least 1 for a non-separable problem to have a solution even with the addition of slack variables. In fact, this reasoning

implies that a budget C allows for at most C misclassifications in the training set. And in a non-separable case, C must be allowed to be large enough so that some choice of $\beta \in \mathfrak{R}^p$ and $\beta_0 \in \mathfrak{R}$ produces a classifier with training error rate no larger than C/N .

In any event, we consider the optimization problem

$$\begin{aligned} & \text{minimize} && \frac{1}{2} \|\beta\|^2 && \text{subject to} && \begin{cases} y_i (\mathbf{x}'_i \beta + \beta_0) + \xi_i \geq 1 & \forall i \\ \text{for some } \xi_i \geq 0 \text{ with } \sum_{i=1}^N \xi_i \leq C \end{cases} \\ & \beta \in \mathfrak{R}^p && && && \\ & \text{and } \beta_0 \in \mathfrak{R} && && && \end{aligned} \tag{143}$$

that can be thought of as generalizing the problem (133). Problem (143) is equivalent to

$$\begin{aligned} & \text{maximize} && M && \text{subject to} && \begin{cases} y_i (\mathbf{x}'_i \mathbf{u} + \beta_0) \geq M(1 - \xi_i) & \forall i \\ \text{for some } \xi_i \geq 0 \text{ with } \sum_{i=1}^N \xi_i \leq C \end{cases} \\ & \mathbf{u} \text{ with } \|\mathbf{u}\| = 1 && && && \\ & \text{and } \beta_0 \in \mathfrak{R} && && && \end{aligned}$$

generalizing the original problem (131). In this latter formulation, the ξ_i represent fractions (of the margin) that a corresponding \mathbf{x}_i is allowed to be on the "wrong side" of its cushion around the classification boundary. $\xi_i > 1$ indicates that not only does \mathbf{x}_i violate its cushion around the surface in \mathfrak{R}^p defined by $\mathbf{x}'\mathbf{u} + \beta_0 = 0$ but that the classifier misclassifies that case.

The ideas and notation of this development are illustrated in Figure 39 for a small $p = 2$ problem.

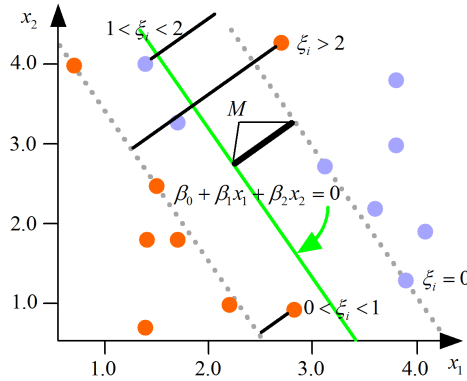


Figure 39: A toy $p = 2$ example illustrating the notation used in non-separable support vector optimization problem statements.

A more convenient version of form (143) is

$$\begin{aligned} & \text{minimize} && \frac{1}{2} \|\beta\|^2 + C^* \sum_{i=1}^N \xi_i && \text{subject to} && \begin{cases} y_i (\mathbf{x}'_i \beta + \beta_0) + \xi_i \geq 1 & \forall i \\ \text{for some } \xi_i \geq 0 \end{cases} \\ & \beta \in \mathfrak{R}^p && && && \\ & \text{and } \beta_0 \in \mathfrak{R} && && && \end{aligned} \tag{144}$$

A nice development on pages 376-378 of Izenman's book provides the following solution to this problem (144) parallel to the development in Section 13.1. Generalizing problem (141) is the dual problem

$$\underset{\alpha \in \mathfrak{R}^N}{\text{maximize}} \quad \mathbf{1}'\alpha - \frac{1}{2}\alpha'\mathbf{H}\alpha \quad \text{subject to} \quad \mathbf{0} \leq \alpha \leq C^*\mathbf{1} \quad \text{and} \quad \alpha'\mathbf{y} = 0 \quad (145)$$

for

$$\mathbf{H}_{N \times N} = (y_i y_j \mathbf{x}_i' \mathbf{x}_j) \quad (146)$$

The constraint $\mathbf{0} \leq \alpha \leq C^*\mathbf{1}$ is known as a "box constraint" and the "feasible region" prescribed in form (145) is the intersection of a hyperplane defined by $\alpha'\mathbf{y} = 0$ and a "box" in the positive orthant. The $C^* = \infty$ version of this reduces to the "hard margin" separable case.

Upon solving problem (145) for α^{opt} , the optimal $\beta \in \mathfrak{R}^p$ is of the form

$$\beta(\alpha^{\text{opt}}) = \sum_{i \in \mathcal{SV}} \alpha_i^{\text{opt}} y_i \mathbf{x}_i \quad (147)$$

for \mathcal{SV} the set of indices of **support vectors** \mathbf{x}_i which have $\alpha_i^{\text{opt}} > 0$. The points with $0 < \alpha_i^{\text{opt}} < C^*$ will lie on the edge of the margin (have $\xi_i = 0$) and the ones with $\alpha_i^{\text{opt}} = C^*$ have $\xi_i > 0$. Any of the support vectors on the edge of the margin (with $0 < \alpha_i^{\text{opt}} < C^*$) may be used to solve for $\beta_0 \in \mathfrak{R}$ as

$$\beta_0(\alpha^{\text{opt}}) = y_i - \mathbf{x}_i' \beta(\alpha^{\text{opt}}) \quad (148)$$

and again, apparently for reasons of numerical stability it is common practice to average values $y_i - \mathbf{x}_i' \beta(\alpha^{\text{opt}})$ for such support vectors in order to evaluate $\beta_0(\alpha^{\text{opt}})$. And here (as in the "hard margin"/no slack case) the margin is related to the coefficients as in display (142).

In this process the constant C^* functions as a regularization/complexity parameter and large C^* in form (144) corresponds to small C in form (143). Identification of a classifier requires only solution of the dual problem (145) and then evaluation of the right hand sides of formulas (147) and (148) to produce linear form (129) and classifier (130). Figure 40 illustrates two different support vector classifiers for a small $p = 2$ problem.

Even when a problem is linearly separable, there may be good reason to use the present formulation with $C^* < \infty$ (and a correspondingly larger margin and more support vectors). Small C^* (large C) corresponds to "low complexity" in choice of a classifier and there are many support vectors contributing to the ultimate form of the classifier. This makes the exact form of the classifier less sensitive to a few key data cases than for large C^* . (If the problem were SEL prediction rather than classification, small C^* would be the "low variance/high bias" case.) Cross-validation can be used in practice to choose an appropriate value for C^* .

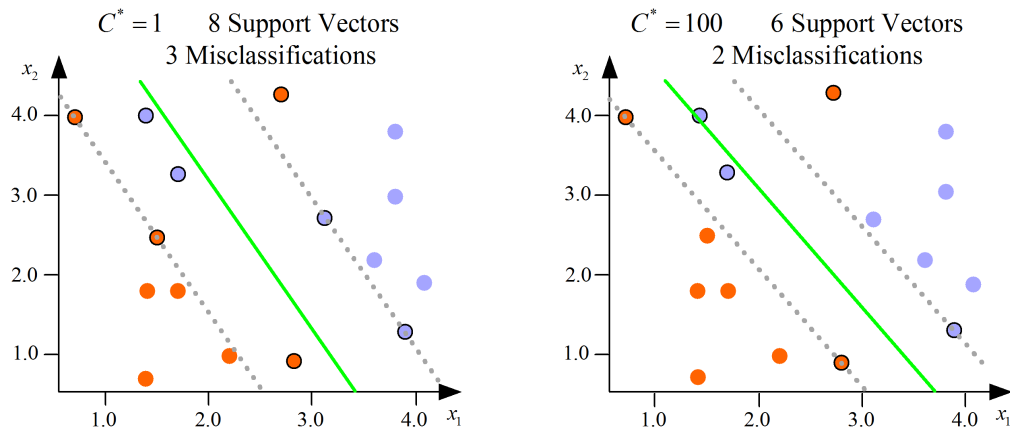


Figure 40: Two support vector classifiers for a small $p = 2$ problem.

13.3 SV Classifiers and Kernels: Support Vector Machines

The form (129) is (of course and by design) linear in the coordinates of \mathbf{x} . A natural generalization of this development would be to consider forms that are linear in some (non-linear) functions of the coordinates of \mathbf{x} . There is nothing really new or special to SV classifiers associated with this possibility if it is applied by simply defining some basis functions $h_m(\mathbf{x})$ and considering form

$$g(\mathbf{x}) = \begin{pmatrix} h_1(\mathbf{x}) \\ h_2(\mathbf{x}) \\ \vdots \\ h_M(\mathbf{x}) \end{pmatrix}' \boldsymbol{\beta} + \beta_0$$

for use as a voting function in a classifier $\text{sign}(g(\mathbf{x}))$. However, the fact that in both linearly separable and linearly non-separable cases, optimal SV classifiers depend upon the training input vectors \mathbf{x}_i only through their inner products (see again displays (140) and (146)) and experience with computing abstract inner products in function spaces using kernel values suggests another way in which one might employ linear forms of nonlinear functions in classification.

13.3.1 Heuristics

Let \mathcal{K} be a non-negative definite kernel and consider the possibility of using functions $\mathcal{K}(\mathbf{x}, \mathbf{x}_1), \mathcal{K}(\mathbf{x}, \mathbf{x}_2), \dots, \mathcal{K}(\mathbf{x}, \mathbf{x}_N)$ to build new (N -dimensional data-dependent) feature vectors

$$\mathbf{k}(\mathbf{x}) = \begin{pmatrix} \mathcal{K}(\mathbf{x}, \mathbf{x}_1) \\ \mathcal{K}(\mathbf{x}, \mathbf{x}_2) \\ \vdots \\ \mathcal{K}(\mathbf{x}, \mathbf{x}_N) \end{pmatrix}$$

for any input vector \mathbf{x} (including the \mathbf{x}_i in the training set) and rather than defining inner products for new feature vectors (for input vectors \mathbf{x} and \mathbf{z}) in terms of \mathfrak{R}^N inner products

$$\langle \mathbf{k}(\mathbf{x}), \mathbf{k}(\mathbf{z}) \rangle = \mathbf{k}(\mathbf{x})' \mathbf{k}(\mathbf{z}) = \sum_{k=1}^N \mathcal{K}(\mathbf{x}, \mathbf{x}_k) \mathcal{K}(\mathbf{z}, \mathbf{x}_k)$$

instead consider using the abstract space inner products of corresponding functions

$$\langle \mathcal{K}(\mathbf{x}, \cdot), \mathcal{K}(\mathbf{z}, \cdot) \rangle_{\mathcal{A}} = \mathcal{K}(\mathbf{x}, \mathbf{z})$$

Then, in place of definition (140) or (146) define

$$\mathbf{H}_{N \times N} = (y_i y_j \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)) \quad (149)$$

and let $\boldsymbol{\alpha}^{\text{opt}}$ solve either problem (141) or (145). With

$$\boldsymbol{\beta}(\boldsymbol{\alpha}^{\text{opt}}) = \sum_{i=1}^N \alpha_i^{\text{opt}} y_i \mathbf{k}(\mathbf{x}_i)$$

as in the developments of the previous sections, we replace the \mathfrak{R}^N inner product of $\boldsymbol{\beta}(\boldsymbol{\alpha}^{\text{opt}})$ and a feature vector $\mathbf{k}(\mathbf{x})$ with

$$\begin{aligned} \left\langle \sum_{i=1}^N \alpha_i^{\text{opt}} y_i \mathcal{K}(\mathbf{x}_i, \cdot), \mathcal{K}(\mathbf{x}, \cdot) \right\rangle_{\mathcal{A}} &= \sum_{i=1}^N \alpha_i^{\text{opt}} y_i \langle \mathcal{K}(\mathbf{x}_i, \cdot), \mathcal{K}(\mathbf{x}, \cdot) \rangle_{\mathcal{A}} \\ &= \sum_{i=1}^N \alpha_i^{\text{opt}} y_i \mathcal{K}(\mathbf{x}, \mathbf{x}_i) \end{aligned}$$

Then for any i for which $\alpha_i^{\text{opt}} > 0$ (an index corresponding to a **support feature vector** in this context) we set

$$\beta_0(\boldsymbol{\alpha}^{\text{opt}}) = y_i - \sum_{j=1}^N \alpha_j^{\text{opt}} y_j \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$$

and have an empirical analogue of voting function (129) (for the kernel case)

$$\hat{g}(\mathbf{x}) = \sum_{i=1}^N \alpha_i^{\text{opt}} y_i \mathcal{K}(\mathbf{x}, \mathbf{x}_i) + \beta_0(\boldsymbol{\alpha}^{\text{opt}}) \quad (150)$$

with corresponding classifier

$$\hat{f}(\mathbf{x}) = \text{sign}(\hat{g}(\mathbf{x})) \quad (151)$$

as an empirical analogue of classifier (130). It remains to argue that this classifier (*developed completely heuristically*) has any kind of rational basis.

13.3.2 A Penalized-Fitting Function-Space Optimization Argument

The heuristic argument for the use of kernels in the SVM context to produce form (150) and classifier (151) is clever enough that some authors simply let it stand on its own as "justification" for using "the kernel trick" of replacing \Re^N inner products of feature vectors with \mathcal{A} inner products of basis functions. Far more satisfying arguments can be made. One is based on an appeal to optimality/regularization considerations provided in a 2002 *Machine Learning* paper of Lin, Wahba, Zhang, and Lee.

Consider \mathcal{A} , an abstract function space³⁸ associated with the non-negative definite kernel \mathcal{K} , and the penalized fitting optimization problem involving the "hinge loss" from Section 1.5.3,

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^N (1 - y_i (\beta_0 + g(\mathbf{x}_i)))_+ + \lambda \frac{1}{2} \|g\|_{\mathcal{A}}^2 && (152) \\ & g \in \mathcal{A} && && \\ & \text{and } \beta_0 \in \Re && && \end{aligned}$$

Dividing the whole optimization criterion in display (152) (hinge loss plus constant times squared \mathcal{A} norm) by N , we see that an empirical version of the expected hinge loss is involved, and can on the basis of the exposition in Section 1.5.3 hope that an element g of \mathcal{A} and value β_0 will be identified in the minimization so that $\beta_0 + g(\mathbf{x})$ is close to the voting function for the optimal 0-1 loss classifier and controls 0-1 loss error rate.

Further, recalling the form (143), the quantity $(1 - y_i (\mathbf{x}'_i \boldsymbol{\beta} + \beta_0))_+$ is the fraction of the margin (M) that input \mathbf{x}_i violates its cushion around the classification boundary hyperplane. (Points on the "right" side of their cushion don't get penalized at all. Ones with $(1 - y_i (\mathbf{x}'_i \boldsymbol{\beta} + \beta_0))_+ = 1$ are on the classification boundary. Ones with $(1 - y_i (\mathbf{x}'_i \boldsymbol{\beta} + \beta_0))_+ > 1$ are points misclassified by the voting function.) The average of such terms is an average fraction (of the margin) violation of the cushion and the optimization seeks to control this, and so the loss really is related to the SV classification ideas.

Then, exactly as will be noted in Section 15, an optimizing $g \in \mathcal{A}$ above must be of the form

$$\begin{aligned} g_{\boldsymbol{\beta}}(\mathbf{x}) &= \sum_{j=1}^N \beta_j \mathcal{K}(\mathbf{x}, \mathbf{x}_j) \\ &= \boldsymbol{\beta}' \mathbf{k}(\mathbf{x}) \end{aligned}$$

so the minimization problem is

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^N (1 - y_i (\beta_0 + \boldsymbol{\beta}' \mathbf{k}(\mathbf{x}_i)))_+ + \lambda \frac{1}{2} \left\| \sum_{j=1}^N \beta_j \mathcal{K}(\mathbf{x}, \mathbf{x}_j) \right\|_{\mathcal{A}}^2 \\ & \boldsymbol{\beta} \in \Re^N && \\ & \text{and } \beta_0 \in \Re && \end{aligned}$$

³⁸To be technically precise, we are talking here about the "Reproducing Kernel Hilbert Space" (RKHS) related to \mathcal{K} . This an abstract function space \mathcal{A} consisting of all linear combinations of slices of the kernel, $\mathcal{K}(\mathbf{x}, \cdot)$ and limits of such linear combinations.

that is,

$$\begin{aligned} & \text{minimize}_{\boldsymbol{\beta} \in \mathfrak{R}^N} \sum_{i=1}^N (1 - y_i (\beta_0 + \boldsymbol{\beta}' \mathbf{k}(\mathbf{x}_i)))_+ + \lambda \frac{1}{2} \boldsymbol{\beta}' \mathbf{K} \boldsymbol{\beta} \\ & \text{and } \beta_0 \in \mathfrak{R} \end{aligned}$$

for

$$\mathbf{K}_{N \times N} = (\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j))$$

the Gram matrix first defined in display (21).

Now this is equivalent to the optimization problem

$$\begin{aligned} & \text{minimize}_{\boldsymbol{\beta} \in \mathfrak{R}^N} \sum_{i=1}^N \xi_i + \lambda \frac{1}{2} \boldsymbol{\beta}' \mathbf{K} \boldsymbol{\beta} \quad \text{subject to} \quad \begin{cases} y_i (\boldsymbol{\beta}' \mathbf{k}(\mathbf{x}_i) + \beta_0) + \xi_i \geq 1 & \forall i \\ \text{for some } \xi_i \geq 0 \end{cases} \\ & \text{and } \beta_0 \in \mathfrak{R} \end{aligned} \tag{153}$$

which for $\mathbf{H}_{N \times N} = (y_i y_j \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j))$ as in (149) has dual problem of the form

$$\text{maximize}_{\boldsymbol{\eta} \in \mathfrak{R}^N} \mathbf{1}' \boldsymbol{\eta} - \frac{1}{2\lambda} \boldsymbol{\eta}' \mathbf{H} \boldsymbol{\eta} \quad \text{subject to } \mathbf{0} \leq \boldsymbol{\eta} \leq \mathbf{1} \text{ and } \boldsymbol{\eta}' \mathbf{y} = 0 \tag{154}$$

or

$$\text{maximize}_{\boldsymbol{\alpha} \in \mathfrak{R}^N} \mathbf{1}' \boldsymbol{\alpha} - \frac{1}{2} \boldsymbol{\alpha}' \left(\frac{1}{\lambda^2} \mathbf{H} \right) \boldsymbol{\alpha} \quad \text{subject to } \mathbf{0} \leq \boldsymbol{\alpha} \leq \lambda \mathbf{1} \text{ and } \boldsymbol{\alpha}' \mathbf{y} = 0 \tag{155}$$

That is, the function space optimization problem (152) has a dual that is the same as for problem (145) *for the choice of* $C^* = \lambda$ *and kernel* $\frac{1}{\lambda^2} \mathcal{K}(\mathbf{x}, \mathbf{z})$ produced by the heuristic argument in Section 13.3.1. Then, if $\boldsymbol{\eta}^{\text{opt}}$ is a solution to (154), Lin *et al.* say that an optimal $\boldsymbol{\beta} \in \mathfrak{R}^N$ is

$$\frac{1}{\lambda} \mathbf{diag}(y_1, \dots, y_N) \boldsymbol{\eta}^{\text{opt}}$$

this producing coefficients to be applied to the functions $\mathcal{K}(\cdot, \mathbf{x}_i)$. On the other hand, the heuristic of Section 13.3.1 prescribes that for $\boldsymbol{\alpha}^{\text{opt}}$ the solution to problem (155) coefficients in the vector

$$\mathbf{diag}(y_1, \dots, y_N) \boldsymbol{\alpha}^{\text{opt}}$$

get applied to the functions $\frac{1}{\lambda^2} \mathcal{K}(\mathbf{x}_i, \cdot)$. Upon recognizing that $\boldsymbol{\eta}^{\text{opt}} = \frac{1}{\lambda} \boldsymbol{\alpha}^{\text{opt}}$ it becomes evident that for the choice of $C^* = \lambda$ and kernel $\frac{1}{\lambda^2} \mathcal{K}$, the heuristic in Section (13.3.1) produces a solution to the optimization problem (152).³⁹

³⁹Differently put, the "kernel trick" of Section (13.3.1) applied to kernel \mathcal{K} with cost parameter C^* solves the present optimization problem applied to kernel $(C^*)^2 \mathcal{K}$ with weighting $\lambda = C^*$ in the problem (152).

13.3.3 A Function-Space-Support-Vector-Classifier Geometry Argument

A different line of argument produces a SVM in a way that connects it to the *geometry* of support vector classification in \mathfrak{R}^p . The basic idea is to recognize that one is mapping input feature vectors to an abstract function space \mathcal{A} via the mapping

$$T(\mathbf{x})(\cdot) = \mathcal{K}(\mathbf{x}, \cdot)$$

and that everything subsequent to this mapping can be done fully honoring the linear space structure. That is, the translation of the support vector classifier argument should be in reference to the geometry of \mathcal{A} . What one is really defining is a classifier with inputs in \mathcal{A} . "Linear classification" in \mathcal{A} is the analogue of support vector classification in \mathfrak{R}^p if one starts from a geometric motivation like that of the support vector classifier development. One seeks a unit vector (now in \mathcal{A}) and a constant so that inner products of the (transformed) data case inputs with the unit vector plus the constant, when multiplied by the y_i , maximize a margin subject to some relaxed constraints.

All this is writable in terms of \mathcal{A} . That is, one wishes to

$$\begin{aligned} & \text{maximize} && M \\ & U \in \mathcal{A} \text{ with } \|U\|_{\mathcal{A}} = 1 \\ & \text{and } \beta_0 \in \mathfrak{R} \\ \text{subject to } & \left\{ \begin{array}{l} y_i (\langle T(\mathbf{x}_i), U \rangle_{\mathcal{A}} + \beta_0) \geq M(1 - \xi_i) \quad \forall i \\ \text{for some } \xi_i \geq 0 \text{ with } \sum_{i=1}^N \xi_i \leq C \end{array} \right. \end{aligned}$$

This is equivalent to the problem

$$\begin{aligned} & \text{minimize} && \frac{1}{2} \|V\|_{\mathcal{A}}^2 \\ & V \in \mathcal{A} \\ & \text{and } \beta_0 \in \mathfrak{R} \\ \text{subject to } & \left\{ \begin{array}{l} y_i (\langle T(\mathbf{x}_i), V \rangle_{\mathcal{A}} + \beta_0) \geq (1 - \xi_i) \quad \forall i \\ \text{for some } \xi_i \geq 0 \text{ with } \sum_{i=1}^N \xi_i \leq C \end{array} \right. \end{aligned}$$

Then either because optimization over all of \mathcal{A} looks too hard, or because some "Representer Theorem" says that it is enough to do so, one might back off from optimization over \mathcal{A} to optimization over the subspace of spanned by the set of N elements $T(\mathbf{x}_i)$. Then writing

$$V = \sum_{i=1}^N \beta_i T(\mathbf{x}_i)$$

so that

$$\frac{1}{2} \|V\|_{\mathcal{A}}^2 = \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \beta_i \beta_j \langle T(\mathbf{x}_i), T(\mathbf{x}_j) \rangle_{\mathcal{A}} = \frac{1}{2} \boldsymbol{\beta}' \mathbf{K} \boldsymbol{\beta}$$

(again, \mathbf{K} is the Gram matrix) the optimization problem becomes

$$\begin{aligned} & \text{minimize} && \frac{1}{2} \boldsymbol{\beta}' \mathbf{K} \boldsymbol{\beta} \\ & \boldsymbol{\beta} \in \mathfrak{R}^N \\ & \text{and } \beta_0 \in \mathfrak{R} \\ \text{subject to } & \left\{ \begin{array}{l} y_i (\boldsymbol{\beta}' \mathbf{K}_i + \beta_0) \geq (1 - \xi_i) \quad \forall i \\ \text{for some } \xi_i \geq 0 \text{ with } \sum_{i=1}^N \xi_i \leq C \end{array} \right. \end{aligned}$$

where \mathbf{K}_i is the i th column of the Gram matrix. For $\boldsymbol{\beta}^{\text{opt}}$ and β_0^{opt} solutions to the optimization problem and

$$V^{\text{opt}} = \sum_{i=1}^N \beta_i^{\text{opt}} T(\mathbf{x}_i)$$

the voting function for the **linear** classifier in \mathcal{A} is (for argument $W \in \mathcal{A}$)

$$\langle W, V^{\text{opt}} \rangle_{\mathcal{A}} + \beta_0^{\text{opt}}$$

The corresponding voting function for the derived **non-linear** classifier on \mathfrak{R}^p is

$$\langle T(\mathbf{x}), V^{\text{opt}} \rangle_{\mathcal{A}} + \beta_0^{\text{opt}} = \sum_{i=1}^N \beta_i^{\text{opt}} \mathcal{K}(\mathbf{x}, \mathbf{x}_i) + \beta_0^{\text{opt}}$$

and one has something very similar to the heuristic application of the "kernel trick." The question is whether it is exactly equivalent to the use of "the trick."

The problem solved by $\boldsymbol{\beta}^{\text{opt}}$ and β_0^{opt} is equivalent for some $\lambda \geq 0$ to

$$\begin{aligned} & \text{minimize}_{\boldsymbol{\beta} \in \mathfrak{R}^N} \quad \frac{1}{2} \boldsymbol{\beta}' \mathbf{K} \boldsymbol{\beta} + \lambda \sum_{i=1}^N \xi_i \quad \text{subject to} \quad \begin{cases} y_i (\boldsymbol{\beta}' \mathbf{K}_i + \beta_0) \geq (1 - \xi_i) & \forall i \\ \text{for some } \xi_i \geq 0 \end{cases} \\ & \text{and } \beta_0 \in \mathfrak{R} \end{aligned} \tag{156}$$

Comparison of display (156) to display (153) and consideration of the argument following statement (153) then shows that there is a choice of C^* for which when using kernel $(1/C^*)^2 \mathcal{K}$ the heuristic/"kernel trick" method produces a solution to the present function-space-support-vector-classifier problem. This is the same circumstance as in the penalized fitting function space optimization argument.⁴⁰

13.3.4 Some Perspective on SVMs

The "kernelizing" of the support vector classifier methodology produces a wide variety of possible classifiers that can be tuned (via cross-validation) over choice of kernel (and any parameters it might have) and C^* or C . As a toy example of what can result from the technology, consider the situation portrayed in Figure 41 with voting functions based on kernels $\mathcal{K}(x, z) = \exp(-\gamma(x - z)^2)$.

In view of the development here and in Section 1.5.3 what is pictured are voting functions that are approximations to the optimal 0-1 loss classifier as linear combinations of the $N = 20$ radial basis functions $\exp(-\gamma(x - x_i)^2)$ plus a constant. The $\gamma = 100$ pictures are understandably more wiggly than the $\gamma = 10$ pictures because of the smaller "bandwidth" of the former basis

⁴⁰The "kernel trick" of Section (13.3.1) applied to kernel \mathcal{K} with cost parameter C^* solves the present geometric optimization problem applied to kernel $(C^*)^2 \mathcal{K}$ with cost parameter C^* .

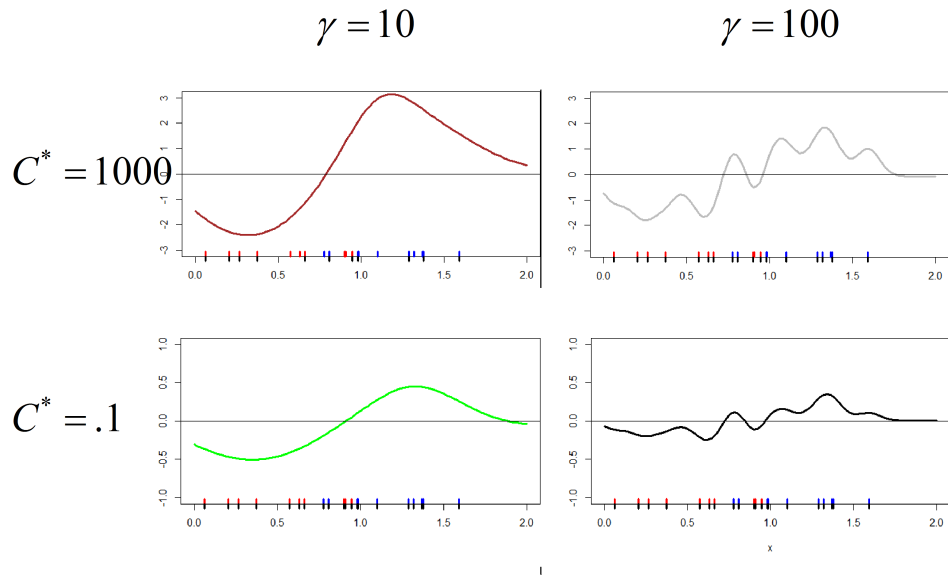


Figure 41: 4 SVM voting functions for a small $p = 1$ example with $N = 20$ cases. Red bars on the rug correspond to $y = -1$ cases and blue bars correspond to $y = 1$ cases. Shown are voting functions based on kernels $\mathcal{K}(x, z) = \exp(-\gamma(x - z)^2)$. The black bars pointing down indicate support "vectors."

functions. The $C^* = 1000$ pictures are closer to being the "hard margin" situation and have fewer training case errors in evidence.

Remember in all this, that SVMs built on a kernel \mathcal{K} will choose voting functions that are linear combinations of the functions $\mathcal{K}(x_i, \cdot)$, slices of the kernel at training case inputs. That fact controls what "shapes" are possible for those voting functions. (In this regard, note that the kernel defined by the ordinary Euclidean inner product, $\mathcal{K}(x, z) = \langle x, z \rangle$, produces linear voting functions and thus linear decision boundaries in \mathbb{R}^p and the special case of ordinary support vector classifiers. It is sometimes called the "linear kernel.")

Finally, it is important to keep in mind that to the extent that SVMs produce good voting functions, those must be equivalent to approximate likelihood ratios. The discussion of Section 1.5.1 still stands.

13.4 Other Support Vector Methods

Several other issues related to the kind of arguments used in the development of SV classifiers are discussed in HTF (and Izenman). One is the matter of multi-class problems. That is, where $\mathcal{G} = \{1, 2, \dots, K\}$ how might one employ machinery of this kind? There are both heuristic and optimality-based methods in the literature.

A heuristic "one-versus-all" (OVA) strategy might be the following. Invent 2-class problems (K of them), the k th based on

$$y_{ki} = \begin{cases} 1 & \text{if } y_i = k \\ -1 & \text{otherwise} \end{cases}$$

Then for (a single) C^* and $k = 1, 2, \dots, K$ solve the (possibly linearly non-separable) 2-class optimization problems to produce functions $\hat{g}_k(\mathbf{x})$ (that would lead to one-versus-all) classifiers $\hat{f}_k(\mathbf{x}) = \text{sign}(\hat{g}_k(\mathbf{x}))$. A possible overall (OVA) classifier is then

$$\hat{f}(\mathbf{x}) = \arg \max_{k \in \mathcal{G}} \hat{g}_k(\mathbf{x})$$

A second heuristic strategy is to develop a voting scheme based on pairwise comparisons. That is, one might invent $\binom{K}{2}$ problems of classifying class l versus class m for $l < m$, choose a single C^* and solve the (possibly linearly non-separable) 2-class optimization problems to produce voting functions $\hat{g}_{lm}(\mathbf{x})$ and corresponding classifiers $\hat{f}_{lm}(\mathbf{x}) = \text{sign}(\hat{g}_{lm}(\mathbf{x}))$. For $m > l$ define $\hat{f}_{ml}(\mathbf{x}) = -\hat{f}_{lm}(\mathbf{x})$ and define an overall "one-versus-one" (OVO) classifier by

$$\hat{f}(\mathbf{x}) = \arg \max_{k \in \mathcal{G}} \left(\sum_{m \neq k} \hat{f}_{km}(\mathbf{x}) \right)$$

or, equivalently

$$\hat{f}(\mathbf{x}) = \arg \max_{k \in \mathcal{G}} \left(\sum_{m \neq k} I[\hat{f}_{km}(\mathbf{x}) = 1] \right)$$

In addition to these fairly ad hoc methods of extending 2-class SVM technology to K -class problems, there are developments that directly address the problem (from an overall optimization point of view). Pages 391-397 of Izenman provide a nice summary of a 2004 paper of Lee, Lin, and Wabha in this direction.

Another type of question related to the support vector material is the extent to which similar methods might be relevant in regression-type prediction problems. As a matter of fact, there are loss functions alternative to squared error or absolute error that lead naturally to the use of the kind of technology needed to produce the SV classifiers. That is, one might consider so called " ϵ insensitive" losses for prediction like

$$L_1^\epsilon(\hat{y}, y) = \max(0, |y - \hat{y}| - \epsilon)$$

or

$$L_2^\epsilon(\hat{y}, y) = \max(0, (y - \hat{y})^2 - \epsilon)$$

and be led to the kind of optimization methods employed in the SVM classification context. See Izenman pages 398-401 in this regard.

14 Prototype and (More on) Nearest Neighbor Methods of Classification

We saw when looking at "linear" methods of classification in Section 12 that these can reduce to classification to the class with fitted mean "closest" in some appropriate sense to an input vector \mathbf{x} . A related notion is to represent classes each by several "prototype" vectors of inputs, and to classify to the class with closest prototype. In this section we have these and related nearest neighbor classifiers in view.

So consider a K -class classification problem (where y takes values in $\mathcal{G} = \{1, 2, \dots, K\}$) and suppose that the coordinates of input \mathbf{x} have been standardized according to training means and standard deviations.

For each class $k = 1, 2, \dots, K$, represent the class by prototypes

$$\mathbf{z}_{k1}, \mathbf{z}_{k2}, \dots, \mathbf{z}_{kR}$$

belonging to \mathfrak{R}^p and consider a classifier/predictor of the form

$$f(\mathbf{x}) = \arg \min_k \min_l \|\mathbf{x} - \mathbf{z}_{kl}\|$$

(that is, one classifies to the class that has a prototype closest to \mathbf{x}).

The most obvious question in using such a rule is "How does one choose the prototypes?" One standard (admittedly ad hoc, but not unreasonable) method is to use the so-called " K -means (clustering) algorithm" (see Section 17.2.1) one class at a time. (The " K " in the name of this algorithm has nothing to do with the number of classes in the present context. In fact, here the " K " naming the clustering algorithm is our present R , the number of prototypes used per class. And the point in applying the algorithm is not so much to see exactly how training vectors aggregate into "homogeneous" groups/clusters as it is to find a few vectors to represent them.)

For $T_k = \{\mathbf{x}_i \text{ with corresponding } y_i = k\}$ an " R -means algorithm might proceed by

1. randomly selecting R different elements from T_k say

$$\mathbf{z}_{k1}^{(1)}, \mathbf{z}_{k2}^{(1)}, \dots, \mathbf{z}_{kR}^{(1)}$$

2. then for $m = 2, 3, \dots$ letting

$$\mathbf{z}_{kl}^{(m)} = \begin{cases} \text{the mean of all } \mathbf{x}_i \in T_k \text{ with} \\ \|\mathbf{x}_i - \mathbf{z}_{kl}^{(m-1)}\| < \|\mathbf{x}_i - \mathbf{z}_{kl'}^{(m-1)}\| \text{ for all } l \neq l' \end{cases}$$

iterating until convergence.

This way of choosing prototypes for class k ignores the "location" of the other classes and the eventual use to which the prototypes will be put. A potential improvement on this is to employ some kind of algorithm (again ad hoc, but

reasonable) that moves prototypes in the direction of training input vectors in their own class and away from training input vectors from other classes. One such method is known by the name "LVQ"/"learning vector quantization." This proceeds as follows.

With a set of prototypes (chosen randomly or from an R -means algorithm or some other way)

$$\mathbf{z}_{kl} \quad k = 1, 2, \dots, K \quad \text{and} \quad l = 1, 2, \dots, R$$

in hand, at each iteration $m = 1, 2, \dots$ for some sequence of "learning rates" $\{\epsilon_m\}$ with $\epsilon_m \geq 0$ and $\epsilon_m \searrow 0$

1. sample an \mathbf{x}_i at random from the training set and find k, l minimizing $\|\mathbf{x}_i - \mathbf{z}_{kl}\|$ (i.e. find the closest prototype \mathbf{z}_{kl})
2. if $y_i = k$ (from 1.), update \mathbf{z}_{kl} as

$$\mathbf{z}_{kl}^{\text{new}} = \mathbf{z}_{kl} + \epsilon_m (\mathbf{x}_i - \mathbf{z}_{kl})$$

and if $y_i \neq k$ (from 1.), update \mathbf{z}_{kl} as

$$\mathbf{z}_{kl}^{\text{new}} = \mathbf{z}_{kl} - \epsilon_m (\mathbf{x}_i - \mathbf{z}_{kl})$$

iterating until convergence.

As early as Section 1.3.3, we considered nearest neighbor methods. Consider here again their use in classification problems. As before, define for each \mathbf{x} the l -neighborhood

$$n_l(\mathbf{x}) = \text{the set of } l \text{ inputs } \mathbf{x}_i \text{ in the training set closest to } \mathbf{x} \text{ in } \mathfrak{R}^p$$

A nearest neighbor method is to classify \mathbf{x} to the class with the largest representation in $n_l(\mathbf{x})$ (possibly breaking ties at random). That is, define

$$\hat{f}(\mathbf{x}) = \arg \max_k \sum_{\mathbf{x}_i \in n_l(\mathbf{x})} I[y_i = k] \quad (157)$$

l is a complexity parameter that might be chosen by cross-validation. Properly implemented this kind of classifier can be highly effective in spite of the curse of dimensionality. This often depends upon clever/appropriate/application-specific choice of feature vectors/functions, definition of appropriate "distance" in order to define "closeness" and the neighborhoods, and appropriate local or global dimension reduction.

A possibility for "local" dimension reduction is this. At $\mathbf{x} \in \mathfrak{R}^p$ one might use regular Euclidean distance to find, say, 50 neighbors of \mathbf{x} in the training inputs to use to identify an appropriate local distance to employ in actually defining the neighborhood $n_l(\mathbf{x})$ to be employed in classifier (157). The following is a DANN (discriminant adaptive nearest neighbor) (squared) metric at $\mathbf{x} \in \mathfrak{R}^p$. Let

$$D^2(\mathbf{z}, \mathbf{x}) = (\mathbf{z} - \mathbf{x})' \mathbf{Q}(\mathbf{z} - \mathbf{x})$$

for

$$\begin{aligned}\mathbf{Q} &= \mathbf{W}^{-\frac{1}{2}} \left(\mathbf{W}^{-\frac{1}{2}} \mathbf{B} \mathbf{W}^{-\frac{1}{2}} + \epsilon \mathbf{I} \right) \mathbf{W}^{-\frac{1}{2}} \\ &= \mathbf{W}^{-\frac{1}{2}} (\mathbf{B}^* + \epsilon \mathbf{I}) \mathbf{W}^{-\frac{1}{2}}\end{aligned}\tag{158}$$

for some $\epsilon > 0$ where \mathbf{W} is a pooled within-class sample covariance matrix

$$\begin{aligned}\mathbf{W} &= \sum_{k=1}^K \hat{\pi}_k \mathbf{W}_k \\ &= \sum_{k=1}^K \hat{\pi}_k \left(\frac{1}{n_k - 1} \sum (\mathbf{x}_i - \bar{\mathbf{x}}_k) (\mathbf{x}_i - \bar{\mathbf{x}}_k)' \right)\end{aligned}$$

($\bar{\mathbf{x}}_k$ is the average \mathbf{x}_i from class k in the 50 used to create the local metric), \mathbf{B} is a weighted between-class covariance matrix of sample means

$$\mathbf{B} = \sum_{k=1}^K \hat{\pi}_k (\bar{\mathbf{x}}_k - \bar{\mathbf{x}}) (\bar{\mathbf{x}}_k - \bar{\mathbf{x}})'$$

($\bar{\mathbf{x}}$ is a (probably weighted) average of the $\bar{\mathbf{x}}_k$) and

$$\mathbf{B}^* = \mathbf{W}^{-\frac{1}{2}} \mathbf{B} \mathbf{W}^{-\frac{1}{2}}$$

Notice that in form (158), the "outside" $\mathbf{W}^{-\frac{1}{2}}$ factors "sphere" ($\mathbf{z} - \mathbf{x}$) differences relative to the within-class covariance structure. \mathbf{B}^* is then the between-class covariance matrix of sphered sample means. Without the $\epsilon \mathbf{I}$, the distance would then discount differences in the directions of the eigenvectors corresponding to large eigenvalues of \mathbf{B}^* (allowing the neighborhood defined in terms of D to be severely elongated in those directions). The effect of adding the $\epsilon \mathbf{I}$ term is to limit this elongation to some degree, preventing \mathbf{x}_i "too far in terms of Euclidean distance from \mathbf{x} " from being included in $n_i(\mathbf{x})$.

A global use of the DANN kind of thinking might be to do the following. At each training input vector $\mathbf{x}_i \in \mathfrak{R}^p$, one might again use regular Euclidean distance to find, say, 50 neighbors and compute a weighted between-class-mean covariance matrix \mathbf{B}_i as above (for that \mathbf{x}_i). These might be averaged to produce

$$\bar{\mathbf{B}} = \frac{1}{N} \sum_{i=1}^N \mathbf{B}_i$$

Then for eigenvalues of $\bar{\mathbf{B}}$, say $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_p \geq 0$ with corresponding (unit) eigenvectors $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_p$ one might do nearest neighbor classification based on the first few features

$$v_j = \mathbf{u}_j' \mathbf{x}$$

and ordinary Euclidean distance. This is a nearest neighbor version of the reduced rank classification idea first met in the discussion of linear classification in Section 12.

15 Reproducing Kernel Hilbert Spaces: Penalized/Regularized and Bayes Prediction

A general framework that unifies many interesting regularized fitting methods is that of reproducing kernel Hilbert spaces (RKHSs). There is a very nice 2012 *Statistics Surveys* paper by Nancy Heckman (related to an older UBC Statistics Department technical report (#216)) entitled "The Theory and Application of Penalized Least Squares Methods or Reproducing Kernel Hilbert Spaces Made Easy," that is the nicest exposition I know about of the connection of this material to splines. Parts of what follows are borrowed shamelessly from her paper. There is also some very helpful stuff in CFZ Section 3.5 and scattered through Izenman about RKHSs.⁴¹

15.1 RKHSs and $p = 1$ Cubic Smoothing Splines

To provide motivation for a somewhat more general discussion, consider again the smoothing spline problem. We consider here the function space

$$\mathcal{A} = \left\{ h : [0, 1] \rightarrow \mathfrak{R} \mid h \text{ and } h' \text{ are absolutely continuous and } \int_0^1 (h''(x))^2 dx < \infty \right\}$$

as a Hilbert space (a linear space with inner product where Cauchy sequences have limits) with inner product

$$\langle f, g \rangle_{\mathcal{A}} \equiv f(0)g(0) + f'(0)g'(0) + \int_0^1 f''(x)g''(x) dx$$

(and corresponding norm $\|h\|_{\mathcal{A}} = \langle h, h \rangle_{\mathcal{A}}^{1/2}$). With this definition of inner product and norm, for $x \in [0, 1]$ the (linear) functional (a mapping $\mathcal{A} \rightarrow \mathfrak{R}$)

$$F_x[f] \equiv f(x)$$

is continuous. Thus the so-called "Riesz representation theorem" says that there is an $R_x \in \mathcal{A}$ such that

$$F_x[f] = \langle R_x, f \rangle_{\mathcal{A}} = f(x) \quad \forall f \in \mathcal{A}$$

(R_x is called "the representer of evaluation at x ." It is in fact possible to verify that for $z \in [0, 1]$)

$$R_x(z) = 1 + xz + R_{1x}(z)$$

for

$$R_{1x}(z) = xz \min(x, z) - \frac{x+z}{2} (\min(x, z))^2 + \frac{1}{3} (\min(x, z))^3$$

does this job.

⁴¹See also "Penalized Splines and Reproducing Kernels" by Pearce and Wand in *The American Statistician* (2006) and "Kernel Methods in Machine Learning" by Hofmann, Scholköpfung, and Smola in *The Annals of Statistics* (2008).

Then the function of two variables defined by

$$R(x, z) \equiv R_x(z)$$

is called the **reproducing kernel** for \mathcal{A} , and \mathcal{A} is called a reproducing kernel Hilbert space (RKHS) (of functions).

Define a linear differential operator on elements of \mathcal{A} (a map from \mathcal{A} to some appropriate function space) by

$$L[f](x) = f''(x)$$

Then the optimization problem solved by the cubic smoothing spline is minimization of

$$\sum_{i=1}^N (y_i - F_{x_i}[h])^2 + \lambda \int_0^1 (L[h](x))^2 dx \quad (159)$$

over choices of $h \in \mathcal{A}$.

It is possible to show that the minimizer of the quantity (159) is necessarily of the form

$$h(x) = \alpha_0 + \alpha_1 x + \sum_{i=1}^N \beta_i R_{1x_i}(x)$$

and that for such h , the criterion (159) is of the form

$$(\mathbf{Y} - \mathbf{T}\boldsymbol{\alpha} - \mathbf{K}\boldsymbol{\beta})'(\mathbf{Y} - \mathbf{T}\boldsymbol{\alpha} - \mathbf{K}\boldsymbol{\beta}) + \lambda \boldsymbol{\beta}' \mathbf{K} \boldsymbol{\beta}$$

for

$$\mathbf{T}_{N \times 2} = (\mathbf{1}, \mathbf{X}), \boldsymbol{\alpha} = \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix}, \text{ and } \mathbf{K} = (R_{1x_i}(x_j))$$

So this has ultimately produced a matrix calculus problem.

15.2 What is Possible Beginning from Linear Functionals and Linear Differential Operators for $p = 1$

For constants $d_i > 0$, functionals F_i , and a linear differential operator L defined for continuous functions $w_k(x)$ by

$$L[h](x) = h^{(m)}(x) + \sum_{k=1}^{m-1} w_k(x) h^{(k)}(x)$$

Heckman considers the minimization of

$$\sum_{i=1}^N d_i (y_i - F_i[h])^2 + \lambda \int_a^b (L[h](x))^2 dx \quad (160)$$

in the space of functions

$$\mathcal{A} = \left\{ h : [a, b] \rightarrow \mathfrak{R} \mid \begin{array}{l} \text{derivatives of } h \text{ up to order } m-1 \text{ are} \\ \text{absolutely continuous and } \int_a^b (h^{(m)}(x))^2 dx < \infty \end{array} \right\}$$

One may adopt an inner product for \mathcal{A} of the form

$$\langle f, g \rangle_{\mathcal{A}} \equiv \sum_{k=0}^{m-1} f^{(k)}(a) g^{(k)}(a) + \int_a^b L[f](x) L[g](x) dx$$

and have a RKHS. The assumption is made that the functionals F_i are continuous and linear, and thus that they are representable as $F_i[h] = \langle f_i, h \rangle_{\mathcal{A}}$ for some $f_i \in \mathcal{A}$. An important special case is that where $F_i[h] = h(x_i)$, but other linear functionals have been used, for example $F_i[h] = \int_a^b H_i(x) h(x) dx$ for known H_i .

The form of the reproducing kernel implied by the choice of this inner product is derivable as follows. First, there is a linearly independent set of functions $\{u_1, \dots, u_m\}$ that is a basis for the subspace of \mathcal{A} consisting of those elements h for which $L[h] = 0$ (the zero function). Call this subspace \mathcal{A}_0 . The so-called Wronskian matrix associated with these functions is then

$$\mathbf{W}_{m \times m}(x) = \left(u_i^{(j-1)}(x) \right)$$

With

$$\mathbf{C} = \left(\mathbf{W}(a) \mathbf{W}(a)' \right)^{-1}$$

let

$$R_0(x, z) = \sum_{i,j} C_{ij} u_i(x) u_j(z)$$

Further, there is a so-called Green's function associated with the operator L , a function $G(x, z)$ such that for all $h \in \mathcal{A}$ satisfying $h^{(k)}(a) = 0$ for $k = 0, 1, \dots, m-1$

$$h(x) = \int_a^b G(x, z) L[h](z) dz$$

Let

$$R_1(x, z) = \int_a^b G(x, t) G(z, t) dt$$

The reproducing kernel associated with the inner product and L is then

$$R(x, z) = R_0(x, z) + R_1(x, z)$$

As it turns out, \mathcal{A}_0 is a RKHS with reproducing kernel R_0 under the inner product $\langle f, g \rangle_0 = \sum_{k=0}^{m-1} f^{(k)}(a) g^{(k)}(a)$. Further, the subspace of \mathcal{A} consisting of those h with $h^{(k)}(a) = 0$ for $k = 0, 1, \dots, m-1$ (call it \mathcal{A}_1) is a RKHS with reproducing kernel $R_1(x, z)$ under the inner product $\langle f, g \rangle_1 = \int_a^b L[f](x) L[g](x) dx$. Every element of \mathcal{A}_0 is perpendicular to every element of \mathcal{A}_1 in \mathcal{A} and every $h \in \mathcal{A}$ can be written uniquely as $h_0 + h_1$ for an $h_0 \in \mathcal{A}_0$ and an $h_1 \in \mathcal{A}_1$.

These facts can be used to show that a minimizer of quantity (160) exists and is of the form

$$h(x) = \sum_{k=1}^m \alpha_k u_k(x) + \sum_{i=1}^N \beta_i R_1(x_i, x)$$

For $h(x)$ of this form, loss plus penalty (160) is of the form

$$(\mathbf{Y} - \mathbf{T}\boldsymbol{\alpha} - \mathbf{K}\boldsymbol{\beta})' \mathbf{D} (\mathbf{Y} - \mathbf{T}\boldsymbol{\alpha} - \mathbf{K}\boldsymbol{\beta}) + \lambda \boldsymbol{\beta}' \mathbf{K} \boldsymbol{\beta}$$

for

$$\mathbf{T} = (F_i[u_j]), \boldsymbol{\alpha} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{pmatrix}, \mathbf{D} = \mathbf{diag}(d_1, \dots, d_m), \text{ and } \mathbf{K} = (F_i[R_1(\cdot, x_j)])$$

and its optimization is a matrix calculus problem. In the important special case where the F_i are function evaluation at x_i , above

$$F_i[u_j] = u_j(x_i) \text{ and } F_i[R_1(\cdot, x_j)] = R_1(x_i, x_j)$$

15.3 What Is Common Beginning Directly From a Kernel

Another way that it is common to make use of RKHSs is to begin with a kernel and its implied inner product (rather than deriving one as appropriate to a particular well-formed optimization problem in function spaces). The 2006 paper of Pearce and Wand in *The American Statistician* is a readable account of this thinking that is more complete than what follows here (and provides some references).

This development begins with C a compact subset of \mathfrak{R}^p and a symmetric kernel function

$$\mathcal{K} : C \times C \rightarrow \mathfrak{R}$$

Ultimately, we will consider as predictors for $\mathbf{x} \in C$ linear combinations of sections of the kernel function, $\sum_{i=1}^N b_i \mathcal{K}(\mathbf{x}, \mathbf{x}_i)$ (where the \mathbf{x}_i are the input vectors in the training set). But to get there in a rational way, and to incorporate use of a complexity penalty into the fitting, we will restrict attention to those kernels that have nice properties. In particular, we require that \mathcal{K} be continuous and non-negative definite.

Then according to what is known as Mercer's Theorem \mathcal{K} may then be written in the form

$$\mathcal{K}(\mathbf{z}, \mathbf{x}) = \sum_{i=1}^{\infty} \gamma_i \phi_i(\mathbf{z}) \phi_i(\mathbf{x}) \quad (161)$$

for linearly independent (in $L_2(C)$) functions $\{\phi_i\}$, and constants $\gamma_i \geq 0$, where the ϕ_i comprise an orthonormal basis for $L_2(C)$ (so any function in

$f \in L_2(C)$ has an expansion in terms of the ϕ_i as $\sum_{i=1}^{\infty} \langle f, \phi_i \rangle_2 \phi_i$ for $\langle f, h \rangle_2 \equiv \int_C f(\mathbf{x}) h(\mathbf{x}) d\mathbf{x}$, and the ones corresponding to positive γ_i may be taken to be continuous on C . Further, the ϕ_i are "eigenfunctions" of the kernel \mathcal{K} corresponding to the "eigenvalues γ_i " in the sense that in $L_2(C)$

$$\int \phi_i(\mathbf{z}) \mathcal{K}(\mathbf{z}, \cdot) d\mathbf{z} = \gamma_i \phi_i(\cdot)$$

Our present interest is in a function space \mathcal{A} (that is a subset of $L_2(C)$ with a different inner product and norm) with members of the form

$$f(\mathbf{x}) = \sum_{i=1}^{\infty} c_i \phi_i(\mathbf{x}) \quad \text{for } c_i \text{ with } \sum_{i=1}^{\infty} \frac{c_i^2}{\gamma_i} < \infty \quad (162)$$

(called the "primal form" of functions in the space). (Notice that all elements of $L_2(C)$ are of the form $f(\mathbf{x}) = \sum_{i=1}^{\infty} c_i \phi_i(\mathbf{x})$ with $\sum_{i=1}^{\infty} c_i^2 < \infty$.) More naturally, our interest centers on

$$f(\mathbf{x}) = \sum_{i=1}^{\infty} b_i \mathcal{K}(\mathbf{x}, \mathbf{z}_i) \quad \text{for some set of } \mathbf{z}_i \quad (163)$$

supposing that the series converges appropriately (called the "dual form" of functions in the space). The former is most useful for producing simple proofs, while the second is most natural for application, since how to obtain the ϕ_i and corresponding γ_i for a given \mathcal{K} is not so obvious. Notice that

$$\begin{aligned} \mathcal{K}(\mathbf{z}, \mathbf{x}) &= \sum_{i=1}^{\infty} \gamma_i \phi_i(\mathbf{z}) \phi_i(\mathbf{x}) \\ &= \sum_{i=1}^{\infty} (\gamma_i \phi_i(\mathbf{x})) \phi_i(\mathbf{z}) \end{aligned}$$

and letting $\gamma_i \phi_i(\mathbf{x}) = c_i(\mathbf{x})$, since $\sum_{i=1}^{\infty} c_i^2(\mathbf{x}) / \gamma_i = \sum_{i=1}^{\infty} \gamma_i \phi_i^2(\mathbf{x}) = \mathcal{K}(\mathbf{x}, \mathbf{x}) < \infty$, the function $\mathcal{K}(\cdot, \mathbf{x})$ is of the form (162), so that we can expect functions of the form (163) with absolutely convergent $\sum_{i=1}^{\infty} b_i$ to be of form (162).

In the space of functions (162), we define an inner product (for our Hilbert space)

$$\left\langle \sum_{i=1}^{\infty} c_i \phi_i, \sum_{i=1}^{\infty} d_i \phi_i \right\rangle_{\mathcal{A}} \equiv \sum_{i=1}^{\infty} \frac{c_i d_i}{\gamma_i}$$

so that

$$\left\| \sum_{i=1}^{\infty} c_i \phi_i \right\|_{\mathcal{A}}^2 \equiv \sum_{i=1}^{\infty} \frac{c_i^2}{\gamma_i}$$

Note then that for $f = \sum_{i=1}^{\infty} c_i \phi_i$ belonging to the Hilbert space \mathcal{A} ,

$$\langle f, \mathcal{K}(\cdot, \mathbf{x}) \rangle_{\mathcal{A}} = \sum_{i=1}^{\infty} \frac{c_i \gamma_i \phi_i(\mathbf{x})}{\gamma_i} = \sum_{i=1}^{\infty} c_i \phi_i(\mathbf{x}) = f(\mathbf{x})$$

and so $\mathcal{K}(\cdot, \mathbf{x})$ is the representer of evaluation at \mathbf{x} . Further,

$$\langle \mathcal{K}(\cdot, \mathbf{z}), \mathcal{K}(\cdot, \mathbf{x}) \rangle_{\mathcal{A}} \equiv \mathcal{K}(\mathbf{z}, \mathbf{x})$$

which is the reproducing property of the RKHS.

Notice also that for two linear combinations of slices of the kernel function at some set of (say, M) inputs $\{\mathbf{z}_i\}$ (two functions in \mathcal{A} represented in dual form)

$$f(\cdot) = \sum_{i=1}^M c_i \mathcal{K}(\cdot, \mathbf{z}_i) \quad \text{and} \quad g(\cdot) = \sum_{i=1}^M d_i \mathcal{K}(\cdot, \mathbf{z}_i)$$

the corresponding \mathcal{A} inner product is

$$\begin{aligned} \langle f, g \rangle_{\mathcal{A}} &= \sum_{i=1}^M c_i \left\langle \mathcal{K}(\cdot, \mathbf{z}_i), \sum_{j=1}^M d_j \mathcal{K}(\cdot, \mathbf{z}_j) \right\rangle_{\mathcal{A}} \\ &= \sum_{i=1}^M \sum_{j=1}^M c_i d_j \mathcal{K}(\mathbf{z}_i, \mathbf{z}_j) \\ &= (c_1, \dots, c_M) (\mathcal{K}(\mathbf{z}_i, \mathbf{z}_j))_{\substack{i=1, \dots, M \\ j=1, \dots, M}} \begin{pmatrix} d_1 \\ \vdots \\ d_M \end{pmatrix} \end{aligned}$$

This is a kind of \Re^M inner product of the coefficient vectors $\mathbf{c}' = (c_1, \dots, c_M)$ and $\mathbf{d}' = (d_1, \dots, d_M)$ defined by the nonnegative definite matrix $(\mathcal{K}(\mathbf{z}_i, \mathbf{z}_j))$. Further, if a random M -vector \mathbf{Y} has covariance matrix $(\mathcal{K}(\mathbf{z}_i, \mathbf{z}_j))$, this is $\text{Cov}(\mathbf{c}'\mathbf{Y}, \mathbf{d}'\mathbf{Y})$. So, in particular, for f of this form $\|f\|_{\mathcal{A}}^2 = \langle f, f \rangle_{\mathcal{A}} = \text{Var}(\mathbf{c}'\mathbf{Y})$.

For applying this material to the fitting of training data, for $\lambda > 0$ and a loss function $L(\hat{y}, y) \geq 0$ define an optimization criterion

$$\text{minimize}_{f \in \mathcal{A}} \left(\sum_{i=1}^N L(f(\mathbf{x}_i), y_i) + \lambda \|f\|_{\mathcal{A}}^2 \right) \quad (164)$$

As it turns out, an optimizer of this criterion must, for the training vectors $\{\mathbf{x}_i\}$, be of the form

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^N b_i \mathcal{K}(\mathbf{x}, \mathbf{x}_i) \quad (165)$$

and the corresponding $\|\hat{f}\|_{\mathcal{A}}^2$ is then

$$\langle \hat{f}, \hat{f} \rangle_{\mathcal{A}} = \sum_{i=1}^N \sum_{j=1}^N b_i b_j \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$$

The criterion (164) is thus

$$\text{minimize}_{\mathbf{b} \in \Re^N} \left(\sum_{i=1}^N L \left(\sum_{j=1}^N b_j \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j), y_i \right) + \lambda \mathbf{b}' (\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)) \mathbf{b} \right) \quad (166)$$

With the Gram matrix $\mathbf{K} = (\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j))$ and $\mathbf{P} = \mathbf{K}^{-1}$ a symmetric generalized inverse of \mathbf{K} and defining

$$L_N(\mathbf{K}\mathbf{b}, \mathbf{Y}) \equiv \sum_{i=1}^N L\left(\sum_{j=1}^N b_j \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j), y_i\right)$$

the optimization criterion (166) is

$$\underset{\mathbf{b} \in \mathfrak{R}^N}{\text{minimize}} (L_N(\mathbf{K}\mathbf{b}, \mathbf{Y}) + \lambda \mathbf{b}' \mathbf{K} \mathbf{b})$$

i.e.

$$\underset{\mathbf{b} \in \mathfrak{R}^N}{\text{minimize}} (L_N(\mathbf{K}\mathbf{b}, \mathbf{Y}) + \lambda \mathbf{b}' \mathbf{K}' \mathbf{P} \mathbf{K} \mathbf{b})$$

i.e.

$$\underset{\mathbf{v} \in C(\mathbf{K})}{\text{minimize}} (L(\mathbf{v}, \mathbf{Y}) + \lambda \mathbf{v}' \mathbf{P} \mathbf{v}) \quad (167)$$

That is, the function space optimization problem (164) reduces to the N -dimensional optimization problem (167). A $\mathbf{v}_\lambda \in C(\mathbf{K})$ (the column space of \mathbf{K}) minimizing $L_N(\mathbf{v}, \mathbf{Y}) + \lambda \mathbf{v}' \mathbf{P} \mathbf{v}$ corresponds to \mathbf{b}_λ minimizing $L_N(\mathbf{K}\mathbf{b}, \mathbf{Y}) + \lambda \mathbf{b}' \mathbf{K} \mathbf{b}$ via

$$\mathbf{K} \mathbf{b}_\lambda = \mathbf{v}_\lambda \quad (168)$$

For the particular special case of squared error loss, $L(\hat{y}, y) = (y - \hat{y})^2$, this development has a very explicit punch line. That is,

$$L_N(\mathbf{K}\mathbf{b}, \mathbf{Y}) + \lambda \mathbf{b}' \mathbf{K} \mathbf{b} = (\mathbf{Y} - \mathbf{K}\mathbf{b})' (\mathbf{Y} - \mathbf{K}\mathbf{b}) + \lambda \mathbf{b}' \mathbf{K} \mathbf{b}$$

Some vector calculus shows that this is minimized over choices of \mathbf{b} by

$$\mathbf{b}_\lambda = (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{Y} \quad (169)$$

and corresponding fitted values are

$$\hat{\mathbf{Y}}_\lambda = \mathbf{v}_\lambda = \mathbf{K} (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{Y}$$

Then using fact (169) under squared error loss, the solution to problem (164) is from expression (165)

$$\hat{f}_\lambda(\mathbf{x}) = \sum_{i=1}^N b_{\lambda i} \mathcal{K}(\mathbf{x}, \mathbf{x}_i) \quad (170)$$

To better understand the nature of (167), consider the eigen decomposition of \mathbf{K} in a case where it is non-singular as

$$\mathbf{K} = \mathbf{U} \mathbf{diag}(\eta_1, \eta_2, \dots, \eta_N) \mathbf{U}'$$

for eigenvalues $\eta_1 \geq \eta_2 \geq \dots \geq \eta_N > 0$, where the eigenvector columns of \mathbf{U} comprise an orthonormal basis for \Re^N . The penalty in form (167) is

$$\begin{aligned}\lambda \mathbf{v}' \mathbf{P} \mathbf{v} &= \lambda \mathbf{v}' \mathbf{U} \mathbf{diag}(1/\eta_1, 1/\eta_2, \dots, 1/\eta_N) \mathbf{U}' \mathbf{v} \\ &= \lambda \sum_{j=1}^N \frac{1}{\eta_j} \langle \mathbf{v}, \mathbf{u}_j \rangle^2\end{aligned}$$

Now (remembering that the \mathbf{u}_j comprise an orthonormal basis for \Re^N)

$$\mathbf{v} = \sum_{j=1}^N \langle \mathbf{v}, \mathbf{u}_j \rangle \mathbf{u}_j \quad \text{and} \quad \|\mathbf{v}\|^2 = \langle \mathbf{v}, \mathbf{v} \rangle = \sum_{j=1}^N \langle \mathbf{v}, \mathbf{u}_j \rangle^2$$

so we see that in choosing a \mathbf{v} to optimize $L_N(\mathbf{v}, \mathbf{Y}) + \lambda \mathbf{v}' \mathbf{P} \mathbf{v}$, we penalize highly those \mathbf{v} with large components in the directions of the late eigenvectors of \mathbf{K} (the ones corresponding to its small eigenvalues) thereby tending to suppress those features of a potential \mathbf{v} .

HTF seem to say that the eigenvalues and eigenvectors of the data-dependent $N \times N$ matrix \mathbf{K} are somehow related respectively to the constants γ_i and functions ϕ_i in the representation (161) of \mathcal{K} as a weighted series of products. That seems hard to understand and (even if true) certainly not obvious.

CFZ provide a result summarizing the most general available version of this development, known as "the representer theorem." It says that if $\Omega : [0, \infty) \rightarrow \Re$ is strictly increasing and $L((\mathbf{x}_1, y_1, h(\mathbf{x}_1)), \dots, (\mathbf{x}_N, y_N, h(\mathbf{x}_N))) \geq 0$ is an arbitrary loss function associated with the prediction of each y_i as $h(\mathbf{x}_i)$, then an $h \in \mathcal{A}$ minimizing

$$L((\mathbf{x}_1, y_1, h(\mathbf{x}_1)), (\mathbf{x}_2, y_2, h(\mathbf{x}_2)), \dots, (\mathbf{x}_N, y_N, h(\mathbf{x}_N))) + \Omega(\|h\|_{\mathcal{A}})$$

has a representation as

$$h(\mathbf{x}) = \sum_{i=1}^N \beta_i \mathcal{K}(\mathbf{x}, \mathbf{x}_i)$$

Further, if $\{\psi_1, \psi_2, \dots, \psi_M\}$ is a set of real-valued functions and the $N \times M$ matrix $(\psi_j(\mathbf{x}_i))$ is of rank M then for $h_0 \in \text{span}\{\psi_1, \psi_2, \dots, \psi_M\}$ and $h_1 \in \mathcal{A}$, an $h = h_0 + h_1$ minimizing

$$L((\mathbf{x}_1, y_1, h(\mathbf{x}_1)), (\mathbf{x}_2, y_2, h(\mathbf{x}_2)), \dots, (\mathbf{x}_N, y_N, h(\mathbf{x}_N))) + \Omega(\|h_1\|_{\mathcal{A}})$$

has a representation as

$$h(\mathbf{x}) = \sum_{i=1}^M \alpha_i \psi_i(\mathbf{x}) + \sum_{i=1}^N \beta_i \mathcal{K}(\mathbf{x}, \mathbf{x}_i)$$

The extra generality provided by this theorem for the squared error loss case treated above is that it provides for linear combinations of the functions $\psi_i(\mathbf{x})$ to be unpenalized in fitting. Then for

$$\mathbf{\Psi}_{N \times M} = (\psi_j(\mathbf{x}_i))$$

and

$$\mathbf{R} = \left(\mathbf{I} - \Psi (\Psi' \Psi)^{-1} \Psi' \right) \mathbf{Y}$$

an optimizing α is $\hat{\alpha} = (\Psi' \Psi)^{-1} \Psi' \mathbf{Y}$ and $\hat{\beta}_\lambda$ optimizes

$$(\mathbf{R} - \mathbf{K}\beta)' (\mathbf{R} - \mathbf{K}\beta) + \lambda \beta' \mathbf{K} \beta$$

and the earlier argument implies that $\hat{\beta}_\lambda = (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{R}$.

15.3.1 Reprise of Some Special Cases

Here we briefly consider special cases of this development, making use of kernel functions introduced as early as Section 1.4.3, beginning with the standard kernel in p dimensions

$$\mathcal{K}(\mathbf{z}, \mathbf{x}) = (1 + \langle \mathbf{z}, \mathbf{x} \rangle)^d$$

For fixed \mathbf{x}_i the basis functions $\mathcal{K}(\mathbf{x}, \mathbf{x}_i)$ are d th order polynomials in the entries of \mathbf{x}_i . So the fitting is in terms of such polynomials. Note that since $\mathcal{K}(\mathbf{z}, \mathbf{x})$ is relatively simple here, there seems to be a good chance of explicitly deriving a representation (161) and perhaps working out all the details of what is above in a very concrete setting.

Another standard kernel function in p dimensions is

$$\mathcal{K}(\mathbf{z}, \mathbf{x}) = \exp\left(-\gamma \|\mathbf{z} - \mathbf{x}\|^2\right)$$

and the basis functions $\mathcal{K}(\mathbf{x}, \mathbf{x}_i)$ are essentially spherically symmetric normal pdfs with mean vectors \mathbf{x}_i . (These are "Gaussian radial basis functions" and for $p = 2$, functions (165) produce prediction surfaces in 3-space that have smooth symmetric "mountains" or "craters" at each \mathbf{x}_i , of elevation or depth relative to the rest of the surface governed by b_i and extent governed by γ .)

Of course, Section 1.4.3 provides a number of insights that enable the creation of a wide variety of kernels beyond the few mentioned here.

Then, for example, the standard development of so-called "support vector classifiers" in a 2-class context with y taking values ± 1 , uses some kernel $\mathcal{K}(\mathbf{z}, \mathbf{x})$ and voting function

$$g(\mathbf{x}) = b_0 + \sum_{i=1}^{\infty} b_i \mathcal{K}(\mathbf{x}, \mathbf{x}_i)$$

in combination with loss

$$L(g(\mathbf{x}), y) = [1 - yg(\mathbf{x})]_+$$

(the sign of $g(\mathbf{x})$ providing the classification associated with \mathbf{x}).

15.3.2 Addendum Regarding the Structures of the Spaces Related to a Kernel

An amplification of some aspects of the basic description provided above for the RKHS corresponding to a kernel $\mathcal{K} : C \times C \rightarrow \mathfrak{R}$ is as follows.

From the representation

$$\mathcal{K}(\mathbf{z}, \mathbf{x}) = \sum_{i=1}^{\infty} \gamma_i \phi_i(\mathbf{z}) \phi_i(\mathbf{x})$$

write

$$\phi_i^* = \sqrt{\gamma_i} \phi_i$$

so that the kernel is

$$\mathcal{K}(\mathbf{z}, \mathbf{x}) = \sum_{i=1}^{\infty} \phi_i^*(\mathbf{z}) \phi_i^*(\mathbf{x}) \quad (171)$$

(Note that considered as functions in $L_2(C)$ the ϕ_i^* are orthogonal, but *not* generally orthonormal, since $\langle \phi_i^*, \phi_i^* \rangle_2 \equiv \int_C \gamma_i \phi_i^2(\mathbf{x}) d\mathbf{x} = \gamma_i$ which is typically not 1.) Representation (171) suggests that one think about the *inner product for inputs provided by the kernel* in terms of a transform of an input vector $\mathbf{x} \in \mathfrak{R}^p$ to an *infinite-dimensional feature vector*

$$\boldsymbol{\phi}^*(\mathbf{x}) = (\phi_1^*(\mathbf{x}), \phi_2^*(\mathbf{x}), \dots)$$

and then "ordinary \mathfrak{R}^∞ inner products" defined on those feature vectors.

The function space \mathcal{A} has members of the (primal) form

$$f(\mathbf{x}) = \sum_{i=1}^{\infty} c_i \phi_i(\mathbf{x}) \quad \text{for } c_i \text{ with } \sum_{i=1}^{\infty} \frac{c_i^2}{\gamma_i} < \infty$$

This is perhaps more naturally

$$f(\mathbf{x}) = \sum_{i=1}^{\infty} c_i \phi_i^*(\mathbf{x}) \quad \text{for } c_i \text{ with } \sum_{i=1}^{\infty} \frac{(c_i \sqrt{\gamma_i})^2}{\gamma_i} = \sum_{i=1}^{\infty} c_i^2 < \infty$$

(Again, all elements of $L_2(C)$ are of the form $f(\mathbf{x}) = \sum_{i=1}^{\infty} c_i \phi_i(\mathbf{x})$ with $\sum_{i=1}^{\infty} c_i^2 < \infty$.) The \mathcal{A} inner product of two *functions* of this primal form has been defined as

$$\begin{aligned} \left\langle \sum_{i=1}^{\infty} c_i \phi_i, \sum_{i=1}^{\infty} d_i \phi_i \right\rangle_{\mathcal{A}} &\equiv \sum_{i=1}^{\infty} \frac{c_i d_i}{\gamma_i} \\ &= \left\langle \sum_{i=1}^{\infty} \frac{c_i}{\sqrt{\gamma_i}} \phi_i^*, \sum_{i=1}^{\infty} \frac{d_i}{\sqrt{\gamma_i}} \phi_i^* \right\rangle_{\mathcal{A}} \\ &= \left\langle \left(\frac{c_1}{\sqrt{\gamma_1}}, \frac{c_2}{\sqrt{\gamma_2}}, \dots \right), \left(\frac{d_1}{\sqrt{\gamma_1}}, \frac{d_2}{\sqrt{\gamma_2}}, \dots \right) \right\rangle_{\mathfrak{R}^\infty} \end{aligned}$$

So, two elements of \mathcal{A} written in terms of the ϕ_i^* (instead of their multiples ϕ_i) say $\sum_{i=1}^{\infty} c_i^* \phi_i^*$ and $\sum_{i=1}^{\infty} d_i^* \phi_i^*$ have \mathcal{A} inner product that is the *ordinary* \mathfrak{R}^{∞} inner product of their vectors of coefficients.

Now consider the function

$$\mathcal{K}(\cdot, \mathbf{x}) = \sum_{i=1}^{\infty} \phi_i^*(\cdot) \phi_i^*(\mathbf{x})$$

For $f = \sum_{i=1}^{\infty} c_i^* \phi_i^* \in \mathcal{A}$,

$$\langle f, \mathcal{K}(\cdot, \mathbf{x}) \rangle_{\mathcal{A}} = \left\langle \sum_{i=1}^{\infty} c_i^* \phi_i^*, \mathcal{K}(\cdot, \mathbf{x}) \right\rangle_{\mathcal{A}} = \sum_{i=1}^{\infty} c_i^* \phi_i^*(\mathbf{x}) = f(\mathbf{x})$$

and (perhaps more clearly than above) indeed $\mathcal{K}(\cdot, \mathbf{x})$ is the representer of evaluation at \mathbf{x} in the function space \mathcal{A} .

15.4 Gaussian Process "Priors," Bayes Predictors, and RKHSs

The RKHS material has an interesting connection to Bayes prediction. It's our purpose here to show that connection. Consider an application of essentially Bayesian thinking to the development of a predictor based the use of a Gaussian process as a more or less non-parametric "prior distribution" for the function (of \mathbf{x}) $E[y|\mathbf{x}]$. That is, for purposes of developing a predictor, suppose that one assumes that

$$y = \eta(\mathbf{x}) + \epsilon$$

where

$$\eta(\mathbf{x}) = \mu(\mathbf{x}) + \gamma(\mathbf{x})$$

$E\epsilon = 0$, $\text{Var}\epsilon = \sigma^2$, the function $\mu(\mathbf{x})$ is known (it could be taken to be identically 0) and plays the role of a prior mean for the function (of \mathbf{x})

$$\eta(\mathbf{x}) = E[y|\mathbf{x}]$$

and (independent of errors ϵ), $\gamma(\mathbf{x})$ is a realization of a mean 0 stationary Gaussian process on \mathfrak{R}^p , this Gaussian process describing the prior uncertainty for $\eta(\mathbf{x})$ around $\mu(\mathbf{x})$. More explicitly, the assumption on $\gamma(\mathbf{x})$ is that $E\gamma(\mathbf{x}) = 0$ and $\text{Var}\gamma(\mathbf{x}) = \tau^2$ for all \mathbf{x} , and for some appropriate (correlation) function ρ , $\text{Cov}(\gamma(\mathbf{x}), \gamma(\mathbf{z})) = \tau^2 \rho(\mathbf{x} - \mathbf{z})$ for all \mathbf{x} and \mathbf{z} ($\rho(\mathbf{0}) = 1$ and the function of two variables $\rho(\mathbf{x} - \mathbf{z})$ must be positive definite). The "Gaussian" assumption is then that for any finite set of elements of \mathfrak{R}^p , say $\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_M$, the vector of corresponding values $\gamma(\mathbf{z}_i)$ is multivariate normal.

There are a number of standard forms that have been suggested for the correlation function ρ . The simplest ones are of a product form, i.e. if ρ_j is a valid one-dimensional correlation function, then the product

$$\rho(\mathbf{x} - \mathbf{z}) = \prod_{j=1}^p \rho_j(x_j - z_j)$$

is a valid correlation function for a Gaussian process on \mathfrak{R}^p . Standard forms for correlation functions in one dimension are $\rho(\Delta) = \exp(-c\Delta^2)$ and $\rho(\Delta) = \exp(-c|\Delta|)$.⁴² The first produces "smoother" realizations than does the second, and in both cases, the constant c governs how fast realizations vary.

One may then consider the joint distribution (conditional on the \mathbf{x}_i and assuming that for the training values y_i the ϵ_i are iid independent of the $\gamma(\mathbf{x}_i)$) of the training output values and a value of $\eta(\mathbf{x})$. From this, one can find the conditional mean for $\eta(\mathbf{x})$ given the training data. To that end, let

$$\boldsymbol{\Sigma}_{N \times N} = (\tau^2 \rho(\mathbf{x}_i - \mathbf{x}_j))_{\substack{i=1,2,\dots,N \\ j=1,2,\dots,N}}$$

Then for a single value of \mathbf{x} ,

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \\ \eta(\mathbf{x}) \end{pmatrix} \sim \text{MVN}_{N+1} \left(\begin{pmatrix} \mu(\mathbf{x}_1) \\ \mu(\mathbf{x}_2) \\ \vdots \\ \mu(\mathbf{x}_N) \\ \mu(\mathbf{x}) \end{pmatrix}, \left(\begin{array}{c|c} (\boldsymbol{\Sigma} + \sigma^2 \mathbf{I}) & \boldsymbol{\Sigma}(\mathbf{x}) \\ \hline \boldsymbol{\Sigma}(\mathbf{x})' & \tau^2 \end{array} \right) \right)$$

for

$$\boldsymbol{\Sigma}(\mathbf{x})_{N \times 1} = \begin{pmatrix} \tau^2 \rho(\mathbf{x} - \mathbf{x}_1) \\ \tau^2 \rho(\mathbf{x} - \mathbf{x}_2) \\ \vdots \\ \tau^2 \rho(\mathbf{x} - \mathbf{x}_N) \end{pmatrix}$$

Then standard multivariate normal theory says that the conditional mean of $\eta(\mathbf{x})$ given \mathbf{Y} is

$$\hat{f}(\mathbf{x}) = \mu(\mathbf{x}) + \boldsymbol{\Sigma}(\mathbf{x})' (\boldsymbol{\Sigma} + \sigma^2 \mathbf{I})^{-1} \begin{pmatrix} y_1 - \mu(\mathbf{x}_1) \\ y_2 - \mu(\mathbf{x}_2) \\ \vdots \\ y_N - \mu(\mathbf{x}_N) \end{pmatrix} \quad (172)$$

Write

$$\mathbf{w}_{N \times 1} = (\boldsymbol{\Sigma} + \sigma^2 \mathbf{I})^{-1} \begin{pmatrix} y_1 - \mu(\mathbf{x}_1) \\ y_2 - \mu(\mathbf{x}_2) \\ \vdots \\ y_N - \mu(\mathbf{x}_N) \end{pmatrix} \quad (173)$$

and then note that form (172) implies that

$$\hat{f}(\mathbf{x}) = \mu(\mathbf{x}) + \sum_{i=1}^N w_i \tau^2 \rho(\mathbf{x} - \mathbf{x}_i) \quad (174)$$

⁴²See Section 1.4.3 for other $p = 1$ bounded non-negative definite functions that can be used to create correlation functions.

and we see that this development ultimately produces $\mu(\mathbf{x})$ plus a linear combination of the "basis functions" $\tau^2\rho(\mathbf{x} - \mathbf{x}_i)$ as a predictor. Remembering that $\tau^2\rho(\mathbf{x} - \mathbf{z})$ must be positive definite and seeing the ultimate form of the predictor, we are reminded of the RKHS material.

In fact, consider the case where $\mu(\mathbf{x}) \equiv 0$. (If one has some non-zero prior mean for $\eta(\mathbf{x})$, arguably that mean function should be subtracted from the raw training outputs before beginning the development of a predictor. At a minimum, output values should probably be centered before attempting development of a predictor.) Compare displays (173) and (174) to displays (169) and (170) for the $\mu(\mathbf{x}) = 0$ case. What is then clear is that the present "Bayes" Gaussian process development of a predictor under squared error loss based on a covariance function $\tau^2\rho(\mathbf{x} - \mathbf{z})$ and error variance σ^2 is equivalent to a RKHS regularized fit of a function to training data based on a kernel $\mathcal{K}(\mathbf{x}, \mathbf{z}) = \tau^2\rho(\mathbf{x} - \mathbf{z})$ and penalty weight $\lambda = \sigma^2$.

16 More on Understanding and Predicting Predictor Performance

There are a variety of theoretical and empirical quantities that might be computed to quantify predictor performance. Those that are empirical and reliably track important theoretical ones might potentially be used to select an effective predictor. We'll here consider some of those (theoretical and empirical) measures. We will do so in the by-now-familiar setting where training data $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)$ are assumed to be iid P and independent of (\mathbf{x}, y) that is also P distributed, and are used to pick a prediction rule $\hat{f}(\mathbf{x})$ to be used under a loss $L(\hat{y}, y) \geq 0$.

What is very easy to think about and compute is the training error

$$\overline{\text{err}} = \frac{1}{N} \sum_{i=1}^N L(\hat{f}(\mathbf{x}_i), y_i)$$

This typically decreases with increased complexity in the form of f , and is no reliable indicator of predictor performance off the training set. Measures of prediction rule performance off the training set must have a theoretical basis (or be somehow based on data held back from the process of prediction rule development).

General loss function versions of (squared error loss) quantities related to $\overline{\text{err}}$ defined in Section ??, are

$$\begin{aligned} \text{Err}(\mathbf{x}) &\equiv \mathbb{E}^T \mathbb{E} \left[L(\hat{f}(\mathbf{x}), y) \mid \mathbf{x} \right] \\ \text{Err}_{\mathcal{T}} &\equiv \mathbb{E}^{(\mathbf{x}, y)} L(\hat{f}(\mathbf{x}), y) \end{aligned} \tag{175}$$

and

$$\text{Err} \equiv \mathbb{E}^{\mathbf{x}} \text{Err}(\mathbf{x}) = \mathbb{E}^T \text{Err}_{\mathcal{T}} \tag{176}$$

A slightly different and semi-empirical version of this expected prediction error (176) is the "in-sample" test error (7.12) of HTF

$$\frac{1}{N} \sum_{i=1}^N \text{Err}(\mathbf{x}_i)$$

16.1 Optimism of the Training Error

Typically, $\overline{\text{err}}$ is less than $\text{Err}_{\mathcal{T}}$. Part of the difference in these is potentially due to the fact that $\text{Err}_{\mathcal{T}}$ is an "extra-sample" error, in that the averaging in (175) is potentially over values \mathbf{x} outside the set of values in the training data. We might consider instead

$$\text{Err}_{\mathcal{T}_{\text{in}}} = \frac{1}{N} \sum_{i=1}^N \mathbb{E}^{y_i^*} L(\hat{f}(\mathbf{x}_i), y_i^*) \quad (177)$$

where the expectations indicated in form (177) are over $y_i^* \sim P_{y|\mathbf{x}=\mathbf{x}_i}$ (the entire training sample used to choose \hat{f} , both inputs and outputs, is being held constant in the averaging in display (177)). The difference

$$\text{op} = \text{Err}_{\mathcal{T}_{\text{in}}} - \overline{\text{err}}$$

is called the "optimism of the training error." HTF use the notation

$$\omega = \mathbb{E}^{\mathbf{Y}} \text{op} = \mathbb{E}^{\mathbf{Y}} (\text{Err}_{\mathcal{T}_{\text{in}}} - \overline{\text{err}}) \quad (178)$$

where the averaging indicated by $\mathbb{E}^{\mathbf{Y}}$ is over the outputs in the training set (using the conditionally independent y_i s, $y_i \sim P_{y|\mathbf{x}=\mathbf{x}_i}$). HTF say that for many losses

$$\omega = \frac{2}{N} \sum_{i=1}^N \text{Cov}^{\mathbf{Y}}(\hat{y}_i, y_i) \quad (179)$$

For example, consider the case of squared error loss. There

$$\begin{aligned} \omega &= \mathbb{E}^{\mathbf{Y}} (\text{Err}_{\mathcal{T}_{\text{in}}} - \overline{\text{err}}) \\ &= \mathbb{E}^{\mathbf{Y}} \mathbb{E}^{\mathbf{Y}^*} \frac{1}{N} \sum_{i=1}^N (y_i^* - \hat{f}(\mathbf{x}_i))^2 - \mathbb{E}^{\mathbf{Y}} \frac{1}{N} \sum_{i=1}^N (y_i - \hat{f}(\mathbf{x}_i))^2 \\ &= \frac{2}{N} \sum_{i=1}^N \left(\mathbb{E}^{\mathbf{Y}} y_i \hat{f}(\mathbf{x}_i) - \mathbb{E}^{\mathbf{Y}} \mathbb{E}^{\mathbf{Y}^*} y_i^* \hat{f}(\mathbf{x}_i) \right) \\ &= \frac{2}{N} \sum_{i=1}^N \mathbb{E}^{\mathbf{Y}} \hat{f}(\mathbf{x}_i) (y_i - \mathbb{E}[y|\mathbf{x}=\mathbf{x}_i]) \\ &= \frac{2}{N} \sum_{i=1}^N \text{Cov}^{\mathbf{Y}}(\hat{y}_i, y_i) \end{aligned}$$

We note that in this context, assuming that given the \mathbf{x}_i in the training data the outputs are uncorrelated and have constant variance σ^2 , by relationship (57)

$$\omega = \frac{2\sigma^2}{N} \text{df}(\widehat{\mathbf{Y}})$$

16.2 C_p , AIC and BIC

The fact that

$$\text{Err}_{\mathbf{T}_{\text{in}}} = \overline{\text{err}} + \text{op}$$

suggests the making of estimates of $\omega = \text{E}^{\mathbf{Y}} \text{op}$ and the use of

$$\overline{\text{err}} + \widehat{\omega} \tag{180}$$

as a guide in model selection. This idea produces consideration of the model selection criteria C_p /AIC and BIC.

16.2.1 C_p and AIC

For the situation of least squares fitting with p predictors or basis functions and squared error loss,

$$\text{df}(\widehat{\mathbf{Y}}) = p = \text{tr}(\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}')$$

so that $\sum_{i=1}^N \text{Cov}^{\mathbf{Y}}(\widehat{y}_i, y_i) = p\sigma^2$. Then, if $\widehat{\sigma^2}$ is an estimated error variance based on a low-bias/high-number-of-predictors fit, a version of quantity (180) suitable for this context is Mallows' C_p

$$C_p \equiv \overline{\text{err}} + \frac{2p}{N} \widehat{\sigma^2}$$

In a more general setting, if one can appropriately evaluate or estimate $\sum_{i=1}^N \text{Cov}^{\mathbf{Y}}(\widehat{y}_i, y_i) = \text{df}(\widehat{\mathbf{Y}}) \sigma^2$, a general version of quantity (180) becomes the Akaike information criterion

$$AIC = \overline{\text{err}} + \frac{2}{N} \sum_{i=1}^N \text{Cov}^{\mathbf{Y}}(\widehat{y}_i, y_i) = \overline{\text{err}} + \frac{2\sigma^2}{N} \text{df}(\widehat{\mathbf{Y}})$$

16.2.2 BIC

For situations where fitting is done by maximum likelihood, the Bayesian Information Criterion of Schwarz is an alternative to AIC. That is, where the joint distribution P produces density $P(y|\boldsymbol{\theta}, \mathbf{x})$ for the conditional distribution of $y|\mathbf{x}$ and $\widehat{\boldsymbol{\theta}}$ is the maximum likelihood estimator of $\boldsymbol{\theta}$, a (maximized) log-likelihood is

$$\text{loglik} = \sum_{i=1}^N \log P(y_i|\widehat{\boldsymbol{\theta}}, \mathbf{x}_i)$$

and the so-called Bayesian information criterion

$$BIC = -2 \log \text{lik} + (\log N) \text{df}(\widehat{\mathbf{Y}})$$

For $y|\mathbf{x}$ normal with variance σ^2 , up to a constant, this is

$$BIC = \frac{N}{\sigma^2} \left[\overline{\text{err}} + \frac{(\log N) \sigma^2}{N} \text{df}(\widehat{\mathbf{Y}}) \right]$$

and after switching 2 for $\log N$, BIC is a multiple of AIC. The replacement of 2 with $\log N$ means that when used to guide model/predictor selections, BIC will typically favor simpler models/predictors than will AIC.

The Bayesian origins of BIC can be developed as follows. Suppose (as in Section 11.1) that M models are under consideration, the m th of which has parameter vector $\boldsymbol{\theta}_m$ and corresponding density for training data

$$f_m(\mathbf{T}|\boldsymbol{\theta}_m)$$

with prior density for $\boldsymbol{\theta}_m$

$$g_m(\boldsymbol{\theta}_m)$$

and prior probability for model m

$$\pi(m)$$

With this structure, the posterior distribution of the model index is

$$\pi(m|\mathbf{T}) \propto \pi(m) \int f_m(\mathbf{T}|\boldsymbol{\theta}_m) g_m(\boldsymbol{\theta}_m) d\boldsymbol{\theta}_m$$

Under 0-1 loss and uniform $\pi(\cdot)$, one wants to choose model m maximizing

$$\int f_m(\mathbf{T}|\boldsymbol{\theta}_m) g_m(\boldsymbol{\theta}_m) d\boldsymbol{\theta}_m = f_m(\mathbf{T}) = \text{the } m\text{th marginal of } \mathbf{T}$$

The so-called Laplace approximation says that

$$\log f_m(\mathbf{T}) \approx \log f_m(\mathbf{T}|\widehat{\boldsymbol{\theta}}_m) - \frac{d_m}{2} \log N + O(1)$$

where d_m is the real dimension of $\boldsymbol{\theta}_m$. Assuming that the marginal of \mathbf{x} doesn't change model-to-model or parameter-to-parameter, $\log f_m(\mathbf{T}|\widehat{\boldsymbol{\theta}}_m)$ is $\log \text{lik} + C_N$, where C_N is a function of only the input values in the training set. Then

$$\begin{aligned} -2 \log f_m(\mathbf{T}) &\approx -2 \log \text{lik} + (\log N) d_m + O(1) - 2C_N \\ &= BIC + O(1) - 2C_N \end{aligned}$$

and (at least approximately) choosing m to maximize $f_m(\mathbf{T})$ is choosing m to minimize BIC.

16.3 Cross-Validation Estimation of Err

K -fold cross-validation is described in Section 1.3.6. One hopes that

$$CV(\hat{f}) = \frac{1}{N} \sum_{i=1}^N L(\hat{f}^{k(i)}(\mathbf{x}_i), y_i)$$

estimates Err. In predictor selection, say where predictor \hat{f} has a complexity parameter α , it is common to look at

$$CV(\hat{f}_\alpha)$$

as a function of α , try to optimize, and then refit (with that α) to the whole training set.

K -fold cross-validation can be expected to estimate Err for

$$"N" = \left(1 - \frac{1}{K}\right) N$$

The question of how cross-validation might be expected to do is thus related to how Err changes with N (the size of the training sample). The statistical folklore is that typically Err decreases monotonically in N approaching some limiting value as N goes to infinity. The "early" (small N) part of the "Err vs N curve" is steep and the "late" part (large N) is relatively flat. If $(1 - 1/K)N$ is large enough that at such size of the training dataset, the curve is flat, then the effectiveness of cross-validation is limited only by the *noise inherent noise in estimating it, and not by the fact that training sets of size $(1 - 1/K)N$ are not of size N* . Operationally, $K = 5$ or 10 seems standard, though as discussed in Section ?? there is recent evidence in favor of using $K = N$, i.e., LOOCV.

HTF say that for many linear fitting methods (that produce $\hat{\mathbf{Y}} = \mathbf{M}\mathbf{Y}$) including least squares projection and cubic smoothing splines, the $N = K$ (leave one out) cross-validation error is (for \hat{f}^i produced by training on $\mathbf{T} - \{(\mathbf{x}_i, y_i)\}$)

$$CV(\hat{f}) = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{f}^i(\mathbf{x}_i))^2 = \frac{1}{N} \sum_{i=1}^N \left(\frac{y_i - \hat{f}(\mathbf{x}_i)}{1 - M_{ii}} \right)^2$$

(for M_{ii} the i th diagonal element of \mathbf{M}). The so-called generalized cross-validation approximation to this is the much more easily computed

$$\begin{aligned} GCV(\hat{f}) &= \frac{1}{N} \sum_{i=1}^N \left(\frac{y_i - \hat{f}(\mathbf{x}_i)}{1 - \text{tr}(\mathbf{M})/N} \right)^2 \\ &= \frac{\overline{\text{err}}}{(1 - \text{tr}(\mathbf{M})/N)^2} \end{aligned}$$

It is worth noting (per HTF Exercise 7.7) that since $1/(1-x)^2 \approx 1+2x$ for x near 0,

$$\begin{aligned} GCV(\hat{f}) &= \frac{1}{N} \sum_{i=1}^N \left(\frac{y_i - \hat{f}(\mathbf{x}_i)}{1 - \text{tr}(\mathbf{M})/N} \right)^2 \\ &\approx \frac{1}{N} \sum_{i=1}^N (y_i - \hat{f}(\mathbf{x}_i))^2 + \frac{2}{N} \text{tr}(\mathbf{M}) \left(\frac{1}{N} \sum_{i=1}^N (y_i - \hat{f}(\mathbf{x}_i))^2 \right) \end{aligned}$$

which is close to AIC, the difference being that here σ^2 is being estimated based on the model being fit, as opposed to being estimated based on a low-bias/large model.

16.4 Bootstrap Estimation of Err

Suppose that the values of the input vectors in the training set are unique. One might make B bootstrap samples of N (random samples with replacement of size N) from the training set \mathbf{T} , say $\mathbf{T}_1^*, \mathbf{T}_2^*, \dots, \mathbf{T}_B^*$, and train on these bootstrap samples to produce predictors, say

$$\text{predictor } \hat{f}^{*b} \text{ based on } \mathbf{T}_b^*$$

Let C^i be the set of indices $b = 1, 2, \dots, B$ for which $(\mathbf{x}_i, y_i) \notin \mathbf{T}_b^*$. A possible bootstrap estimate of Err is then

$$\widehat{\text{Err}}^{(1)} \equiv \frac{1}{N} \sum_{i=1}^N \left[\frac{1}{|C^i|} \sum_{b \in C^i} L(\hat{f}^{*b}(\mathbf{x}_i), y_i) \right]$$

It's not completely clear what to make of this. For one thing, the \mathbf{T}_b^* rarely have N distinct elements. In fact, the expected number of distinct cases in a bootstrap sample for N of any appreciable size is about $.632N$. So roughly speaking, we might expect $\widehat{\text{Err}}^{(1)}$ to estimate Err at $.632N$, not at N . So unless Err as a function of training set size is fairly flat to the right of $.632N$, one might expect substantial positive bias in it as an estimate of Err (at N).

HTF argue for

$$\widehat{\text{Err}}^{(.632)} \equiv .368 \overline{\text{err}} + .632 \widehat{\text{Err}}^{(1)}$$

as a first order correction on the biased bootstrap estimate, but admit that this is not perfect either, and propose a more complicated fix (that they call $\widehat{\text{Err}}^{(.632+)}$) for classification problems.

Part V

Unsupervised Learning Methods

17 Some Methods of Unsupervised Learning

As we said in Section 1.1, "supervised learning" is basically prediction of y belonging to \mathfrak{R} or some finite index set from a p -dimensional \mathbf{x} with coordinates each individually in \mathfrak{R} or some finite index set, using training data pairs

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)$$

to create an effective prediction rule

$$\hat{y} = \hat{f}(\mathbf{x})$$

This is one kind of discovery and exploitation of structure in the training data.

As we also said in Section 1.1, "unsupervised learning" is discovery and quantification of structure in

$$\mathbf{X}_{N \times p} = \begin{pmatrix} \mathbf{x}'_1 \\ \mathbf{x}'_2 \\ \vdots \\ \mathbf{x}'_N \end{pmatrix}$$

without reference to some particular coordinate of a p -dimensional \mathbf{x} as an object of prediction. There are a number of versions of this problem in Ch 14 of HTF that we will outline here.

17.1 Association Rules/Market Basket Analysis

Suppose that one is presented with a database representing N transactions, each of which may or may not include each one of items

$$s_1, s_2, \dots, s_p$$

so that one could think of \mathbf{x} taking values in $\{0, 1\}^p$, $x_j = 1$ indicating presence of item j in the transaction. For two disjoint sets of items

$$\mathcal{S}_1 = \{s_{11}, s_{12}, \dots, s_{1k_1}\} \quad \text{and} \quad \mathcal{S}_2 = \{s_{21}, s_{22}, \dots, s_{2k_2}\}$$

consider transactions that

1. include all items in \mathcal{S}_1 ,
2. include all items in \mathcal{S}_2 , or
3. include all items in $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$.

In applications of this formalism to "market-basket analysis" it is common to call \mathcal{S} , \mathcal{S}_1 , and \mathcal{S}_2 **item sets** and the statement

"the transaction includes all of both item set \mathcal{S}_1 and item set \mathcal{S}_2 "

a **conjunctive rule**. It is then common to further talk about **association rules** of the form

$$\mathcal{S}_1 \implies \mathcal{S}_2 \quad (181)$$

and to consider quantitative measures associated with them. In framework (181), \mathcal{S}_1 is called the **antecedent** and \mathcal{S}_2 is called the **consequent** in the rule.

Define indicator variables

$$I_{ij} = I[\text{transaction } i \text{ includes all of item set } \mathcal{S}_j]$$

for $i = 1, \dots, N$ and $j = 1, 2$. For the association rule $\mathcal{S}_1 \implies \mathcal{S}_2$,

1. the **support** of the rule (also the support of the item set \mathcal{S}) is

$$\frac{1}{N} \sum_{i=1}^N I_{i1} I_{i2}$$

(the relative frequency with which the full item set is seen in the database/training cases),

2. the **confidence** or **predictability** of the rule is

$$\frac{\sum_{i=1}^N I_{i1} I_{i2}}{\sum_{i=1}^N I_{i1}}$$

(the relative frequency with which the full item set \mathcal{S} is seen in the training cases that exhibit the smaller item set \mathcal{S}_1),

3. the **"expected confidence"** of the rule is

$$\frac{1}{N} \sum_{i=1}^N I_{i2}$$

(the relative frequency with which item set \mathcal{S}_2 is seen in the training cases), and

4. the **lift** of the rule is

$$\frac{\text{confidence}}{\text{expected confidence}} = \frac{N \sum_{i=1}^N I_{i1} I_{i2}}{\left(\sum_{i=1}^N I_{i1}\right) \left(\sum_{i=1}^N I_{i2}\right)}$$

(a measure of association).

If one thinks of the cases in the training set as a random sample from some distribution on item sets (equivalently, a distribution for \mathbf{x}), lets I_1 stand for the event that all items in \mathcal{S}_1 are in the set, I_2 stand for the event that all items in \mathcal{S}_2 are in the set, and I stand for the event that all items in $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$ are in the set, then

1. the support of the rule is an estimate of $P(I)$,
2. the confidence is an estimate of $P(I_2|I_1)$,
3. the expected confidence is an estimate of $P(I_2)$,
4. and the lift is an estimate of the ratio $P(I_1 \text{ and } I_2) / (P(I_1) \cdot P(I_2))$.

The basic thinking about association rules seems to be that usually (but perhaps not always) one wants rules with large support (so that the estimates can be reasonably expected to be reliable). Further, one then wants large confidence or lift, as these indicate that the corresponding rule will be useful in terms of understanding how the coordinates of \mathbf{x} (presence or absence of various items) are related in the database/training data. Apparently, standard practice is to identify a large number of promising item sets and association rules, and make a database of association rules that can be queried in searches like:

"Find all rules in which YYY is the consequent that have confidence over 70% and support more than 1%."

Basic questions that we have to this point not addressed are where one gets appropriate item sets \mathcal{S} and how one uses them to produce (\mathcal{S}_1 and \mathcal{S}_2 and) corresponding association rules. In answer to the second of these questions, one might say "consider all $2^{|\mathcal{S}|-2}$ association rules that can be associated with a given item set." But what then are "interesting" item sets \mathcal{S} or how does one find a potentially useful set of such? We proceed to briefly consider these issues.

17.1.1 The "Apriori Algorithm" and Use of its Output

One standard way of generating item sets (to process into association rules) is to use the so-called "apriori algorithm." This produces all item sets \mathcal{S} of support at least t . (These can then be examined to find potentially interesting association rules by breaking them into two pieces \mathcal{S}_1 and \mathcal{S}_2).

This operates as follows.

1. Pass through all p items

$$s_1, s_2, \dots, s_p$$

identifying those s_j that individually have support/prevalence

$$\frac{1}{N} \cdot \# \{i \mid x_{ij} = 1\}$$

at least t and place them in the set

$$\mathcal{S}_1^t = \{\text{item sets of size 1 with support at least } t\}$$

2. For each $s_j \in \mathcal{S}_1^t$ check to see which two-element item sets

$$\{s_j, s_{j'}\}_{j' \neq j \text{ and } s_{j'} \in \mathcal{S}_1^t}$$

have support/prevalence

$$\frac{1}{N} \cdot \#\{i \mid x_{ij}x_{ij'} = 1\}$$

at least t and place them in the set

$$\mathcal{S}_2^t = \{\text{item sets of size 2 with support at least } t\}$$

\vdots

m . For each $\left\{ \overbrace{s_j, s_{j'}, \dots}^{m-1 \text{ entries}} \right\} \in \mathcal{S}_{m-1}^t$ check to see which m -element item sets

$$\{s_j, s_{j'}, \dots\} \cup \{s_{j^*}\} \text{ for } j^* \notin \{j, j', \dots\} \text{ and } s_{j^*} \in \mathcal{S}_1^t$$

have support/prevalence

$$\frac{1}{N} \cdot \#\{i \mid x_{ij}x_{ij'} \cdots x_{ij^*} = 1\}$$

at least t and place them in the set

$$\mathcal{S}_m^t = \{\text{item sets of size } m \text{ with support at least } t\}$$

This algorithm terminates when at some stage m the set \mathcal{S}_m^t is empty. Then a sensible set of item sets (to consider for making association rules) is $\mathcal{S}^t = \cup \mathcal{S}_m^t$, the set of all item sets with prevalence in the training data of at least t . Apparently for commercial databases of "typical size," unless t is very small it is feasible to use this algorithm to find \mathcal{S}^t . It is also possible to use a variant of the apriori algorithm to find all association rules based on item sets in \mathcal{S}^t with confidence at least c . This then produces a database of association rules that can be queried by a user wishing to identify useful structure in the database/training dataset.

In a more statistical vein, one can adopt from \mathcal{S}^t some consequent of interest $\mathcal{S}^{**} = \{s_1^{**}, s_2^{**}, \dots, s_l^{**}\}$ and consider modeling of the binary variable

$$I[\text{all items in } \mathcal{S}^{**} \text{ are in a transaction}] = \prod_{j \text{ s.t. } s_j \in \mathcal{S}^{**}} x_j$$

on the basis of some non-overlapping set of variables related to an antecedent \mathcal{S}^* (disjoint from \mathcal{S}^{**} belonging to \mathcal{S}^t). For example, a natural possibility is to use logistic regression based on the set of variables x_j with $s_j \in \mathcal{S}^*$ to look for items (or sets of items if products of these indicators are employed) that are associated with "large" (or "increased") probabilities of the consequent.

17.2 Clustering

Typically (but not always) the object in "clustering" is to find natural groups of rows or columns of

$$\mathbf{X}_{N \times p} = \begin{pmatrix} \mathbf{x}'_1 \\ \mathbf{x}'_2 \\ \vdots \\ \mathbf{x}'_N \end{pmatrix}$$

(in some contexts one may want to somehow find homogenous "blocks" in a properly rearranged \mathbf{X}). Sometimes all columns of \mathbf{X} represent values of continuous variables (so that ordinary arithmetic applied to all its elements is meaningful). But sometimes some columns correspond to ordinal or even categorical variables. In light of all this, we will let \mathbf{x}_i $i = 1, 2, \dots, r$ stand for "items" to be clustered (that might be rows or columns of \mathbf{X}) with entries that need not necessarily be continuous variables.

In developing and describing clustering methods, it is often useful to have a dissimilarity measure $d(\mathbf{x}, \mathbf{z})$ that (at least for the items to be clustered and perhaps for other possible items) quantifies how "unlike" items are. This measure is usually chosen to satisfy

1. $d(\mathbf{x}, \mathbf{z}) \geq 0 \forall \mathbf{x}, \mathbf{z}$
2. $d(\mathbf{x}, \mathbf{x}) = 0 \forall \mathbf{x}$, and
3. $d(\mathbf{x}, \mathbf{z}) = d(\mathbf{z}, \mathbf{x}) \forall \mathbf{x}, \mathbf{z}$.

It may be chosen to further satisfy

4. $d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{w}) + d(\mathbf{z}, \mathbf{w}) \forall \mathbf{x}, \mathbf{z}$, and \mathbf{w} , or
- 4'. $d(\mathbf{x}, \mathbf{z}) \leq \max[d(\mathbf{x}, \mathbf{w}), d(\mathbf{z}, \mathbf{w})] \forall \mathbf{x}, \mathbf{z}$, and \mathbf{w} .

Where 1-4 hold, d is a "metric." Where 1-3 hold and the stronger condition 4' holds, d is an "ultrametric."

In a case where one is clustering rows of \mathbf{X} and each column of \mathbf{X} contains values of a continuous variable, a squared Euclidean distance is a natural choice for a dissimilarity measure

$$d(\mathbf{x}_i, \mathbf{x}_{i'}) = \|\mathbf{x}_i - \mathbf{x}_{i'}\|^2 = \sum_{j=1}^p (x_{ij} - x_{i'j})^2$$

In a case where one is clustering columns of \mathbf{X} and each column of \mathbf{X} contains values of a continuous variable, with $r_{jj'}$ the sample correlation between values in columns j and j' , a plausible dissimilarity measure is

$$d(\mathbf{x}_j, \mathbf{x}_{j'}) = 1 - |r_{jj'}|$$

When dissimilarities between r items are organized into a (non-negative symmetric) $r \times r$ matrix

$$\mathbf{D} = (d_{ij}) = (d(\mathbf{x}_i, \mathbf{x}_j))$$

with 0s down its diagonal, the terminology "proximity matrix" is often used. For some clustering algorithms and for some purposes, the proximity matrix encodes all one needs to know about the items to do clustering. One seeks a partition of the index set $\{1, 2, \dots, r\}$ into subsets such that the d_{ij} for indices within a subset are small (and the d_{ij} for indices i and j from different subsets are large).

17.2.1 Partitioning Methods ("Centroid"-Based Methods)

By far the most commonly used clustering methods are based on partitioning related to "centroids," particularly the so called " K -means" clustering algorithm for the rows of \mathbf{X} in cases where the columns contain values of continuous variables x_j (for which arithmetic averaging makes sense).⁴³

The algorithm begins with some set of K distinct "centers" $\mathbf{c}_1^0, \mathbf{c}_2^0, \dots, \mathbf{c}_K^0$. They might, for example, be a random selection of the rows of \mathbf{X} (subject to the constraint that they are distinct). One then assigns each \mathbf{x}_i to that center $\mathbf{c}_{k^0(i)}^0$ minimizing

$$d(\mathbf{x}_i, \mathbf{c}_l^0)$$

over choice of l (creating K clusters around the centers) and replaces all of the \mathbf{c}_k^0 with the corresponding cluster means

$$\mathbf{c}_k^1 = \frac{1}{\# \text{ of } i \text{ with } k^0(i) = k} \sum I[k^0(i) = k] \mathbf{x}_i$$

At stage m with all \mathbf{c}_k^{m-1} available, one then assigns each \mathbf{x}_i to that center $\mathbf{c}_{k^{m-1}(i)}^{m-1}$ minimizing

$$d(\mathbf{x}_i, \mathbf{c}_l^{m-1})$$

over choice of l (creating K clusters around the centers) and replaces all of the \mathbf{c}_k^{m-1} with the corresponding cluster means

$$\mathbf{c}_k^m = \frac{1}{\# \text{ of } i \text{ with } k^{m-1}(i) = k} \sum I[k^{m-1}(i) = k] \mathbf{x}_i$$

This iteration goes on to convergence. One compares multiple random starts for a given K (and then minimum values found for each K) in terms of

$$\text{Total Within-Cluster Dissimilarity}(K) = \sum_{k=1}^K \sum_{\mathbf{x}_i \text{ in cluster } k} d(\mathbf{x}_i, \mathbf{c}_k)$$

⁴³In this context, a natural choice of $d(\mathbf{x}, \mathbf{z})$ is $\|\mathbf{x} - \mathbf{z}\|^2$. A fancier option might be built on squared Mahalanobis distance, $(\mathbf{x} - \mathbf{z})' \mathbf{Q} (\mathbf{x} - \mathbf{z})$ for some non-negative definite \mathbf{Q} .

for $\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_K$ the final means produced by the iterations.⁴⁴ One may then consider the (monotone) sequence of Total Within-Cluster Dissimilarities and try to identify a value K beyond which there seem to be diminishing returns for increased K .

A more general version of this algorithm (that might be termed a K -medoid algorithm) doesn't require that the entries of the \mathbf{x}_i be values of continuous variables, but (since it is then unclear that one can even evaluate, let alone find a general minimizer of, $d(\mathbf{x}_i, \cdot)$) restricts the "centers" to be original items. This algorithm begins with some set of K distinct "medoids" $\mathbf{c}_1^0, \mathbf{c}_2^0, \dots, \mathbf{c}_K^0$ that are a random selection from the r items \mathbf{x}_i (subject to the constraint that they are distinct). One then assigns each \mathbf{x}_i to that medoid $\mathbf{c}_{k^0(i)}^0$ minimizing

$$d(\mathbf{x}_i, \mathbf{c}_l^0)$$

over choice of l (creating K clusters associated with the medoids) and replaces all of the \mathbf{c}_k^0 with \mathbf{c}_k^1 the corresponding minimizers over the $\mathbf{x}_{i'}$ belonging to cluster k of the sums

$$\sum_{i \text{ with } k^0(i)=k} d(\mathbf{x}_i, \mathbf{x}_{i'})$$

At stage m with all \mathbf{c}_k^{m-1} available, one then assigns each \mathbf{x}_i to that medoid $\mathbf{c}_{k^{m-1}(i)}^{m-1}$ minimizing

$$d(\mathbf{x}_i, \mathbf{c}_l^{m-1})$$

over choice of l (creating K clusters around the medoids) and replaces all of the \mathbf{c}_k^{m-1} with \mathbf{c}_k^m the corresponding minimizers over the $\mathbf{x}_{i'}$ belonging to cluster k of the sums

$$\sum_{i \text{ with } k^{m-1}(i)=k} d(\mathbf{x}_i, \mathbf{x}_{i'})$$

This iteration goes on to convergence. One compares multiple random starts for a given K (and then minimum values found for each K) in terms of

$$\sum_{k=1}^K \sum_{\mathbf{x}_i \text{ in cluster } k} d(\mathbf{x}_i, \mathbf{c}_k)$$

for $\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_K$ the final medoids produced by the iterations.

17.2.2 Hierarchical Methods

To apply a hierarchical clustering method, one must first choose a method of using dissimilarities for items to define dissimilarities for clusters. Three common (and somewhat obvious) possibilities in this regard are as follows. For C_1 and C_2 different elements of a partition of the set of items, or equivalently their r indices, one might define dissimilarity of C_1 and C_2 as

⁴⁴For a squared Euclidean distance d , this is a total squared distance of \mathbf{x}_i s to their corresponding cluster means.

1. $D(C_1, C_2) = \min \{d_{ij} | i \in C_1 \text{ and } j \in C_2\}$ (this is the "single linkage" or "nearest neighbor" choice),
2. $D(C_1, C_2) = \max \{d_{ij} | i \in C_1 \text{ and } j \in C_2\}$ (this is the "complete linkage" choice), or
3. $D(C_1, C_2) = \frac{1}{\#C_1 \cdot \#C_2} \sum_{i \in C_1, j \in C_2} d_{ij}$ (this is the "average linkage" choice).

There are both agglomerative/bottom-up methods and divisive/top-down methods of hierarchical clustering. An agglomerative hierarchical clustering algorithm operates as follows. One begins with every item $\mathbf{x}_i, i = 1, 2, \dots, r$ functioning as a singleton cluster. Then one finds the minimum d_{ij} for $i \neq j$ and puts the corresponding two items into a single cluster (of size 2). Then when one is at a stage where there are m clusters, one finds the two clusters with minimum dissimilarity and merges them to make a single cluster, leaving $m - 1$ clusters overall. This continues until there is only a single cluster. The sequence of r different clusterings (with r through 1 clusters) serves as a menu of potentially interesting solutions to the clustering problem. These are often displayed in the form of a dendrogram, where cutting the dendrogram at a given level picks out one of the (increasingly coarse as the level rises) clusterings. Those items clustered together "deep" in the tree/dendrogram are presumably interpreted to be potentially "more alike" than ones clustered together only at a high level.

A divisive hierarchical algorithm operates as follows. Starting with a single "cluster" consisting of all items, one finds the maximum d_{ij} and uses the two corresponding items as seeds for two clusters. One then assigns each \mathbf{x}_l for $l \neq i$ and $l \neq j$ to the cluster represented by \mathbf{x}_i if

$$d(\mathbf{x}_i, \mathbf{x}_l) < d(\mathbf{x}_j, \mathbf{x}_l)$$

and to the cluster represented \mathbf{x}_j otherwise. When one is at a stage where there are m clusters, one identifies the cluster with largest d_{ij} corresponding to a pair of elements in the cluster, splitting it using the method applied to split the original "single large cluster" (to produce an $(m + 1)$ -cluster clustering). This, like the agglomerative algorithm, produces a sequence of r different clusterings (with 1 through r clusters) that serves as a menu of potentially interesting solutions to the clustering problem. And like the sequence produced by the agglomerative algorithm, this sequence can be represented using a dendrogram.

One may modify either the agglomerative or divisive algorithms by fixing a threshold $t > 0$ for use in deciding whether or not to merge two clusters or to split a cluster. The agglomerative version would terminate when all pairs of existing clusters have dissimilarities more than t . The divisive version would terminate when all dissimilarities for pairs of items in all clusters are below t . Fairly obviously, employing a threshold has the potential to shorten the menu of clusterings produced by either of the methods to include less than r clusterings. (Obviously, thresholding the agglomerative method cuts off the top of the corresponding full dendrogram, and thresholding the divisive method cuts off the bottom of the corresponding full dendrogram.)

17.2.3 (Mixture) Model-Based Methods

A completely different approach to clustering into K clusters is based on use of mixture models. That is, for purposes of producing a clustering, one might consider acting as if items $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_r$ are realizations of r iid random vectors with parametric marginal density

$$q(\mathbf{x}|\boldsymbol{\pi}, \boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_K) = \sum_{k=1}^K \pi_k p(\mathbf{x}|\boldsymbol{\theta}_k) \quad (182)$$

for probabilities $\pi_k > 0$ with $\sum_{k=1}^K \pi_k = 1$, a fixed parametric density $p(\mathbf{x}|\boldsymbol{\theta})$, and parameters $\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_K$. (Without further restrictions the family of mixture distributions specified by density (182) is not identifiable, but we'll ignore that fact for the moment.)

A useful way to think about this formalism is in terms of a K -class classification model where values of y are latent/unobserved/completely fictitious. This produces density (182) as the marginal density of \mathbf{x} . Further, in the model including a latent y

$$P[y = k|\mathbf{x}] = \frac{\pi_k p(\mathbf{x}|\boldsymbol{\theta}_k)}{\sum_{k=1}^K \pi_k p(\mathbf{x}|\boldsymbol{\theta}_k)}$$

is the (Bayes posterior) probability that \mathbf{x} was generated by component k of the mixture. It then would make sense to define cluster k to be the set of \mathbf{x}_i for which

$$k = \arg \max_l \frac{\pi_l p(\mathbf{x}_i|\boldsymbol{\theta}_l)}{\sum_{k=1}^K \pi_k p(\mathbf{x}_i|\boldsymbol{\theta}_k)} = \arg \max_l \pi_l p(\mathbf{x}_i|\boldsymbol{\theta}_l)$$

This is the set of \mathbf{x}_i that would be classified to class k by the optimal (Bayes) classifier.

In practice, $\boldsymbol{\pi}, \boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_K$ must be estimated and estimates used in place of parameters in defining clusters. That is, an implementable clustering method is to define cluster k (say, C_k) to be

$$C_k = \left\{ \mathbf{x}_i | k = \arg \max_l \hat{\pi}_l p(\mathbf{x}_i|\hat{\boldsymbol{\theta}}_l) \right\} \quad (183)$$

Given the lack of identifiability in the unrestricted mixture model, it might appear that prescription (183) could be problematic. But such is not really the case. While the likelihood

$$L(\boldsymbol{\pi}, \boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_K) = \prod_{i=1}^r q(\mathbf{x}_i|\boldsymbol{\pi}, \boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_K)$$

will have multiple maxima, using any maximizer for an estimate of the parameter vector will produce the same set of clusters (183). It is common to employ the "EM algorithm" in the maximization of $L(\boldsymbol{\pi}, \boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_K)$ (the finding of one of many maximizers) and to include details of that algorithm in expositions of model-based clustering. However, strictly speaking, that algorithm is not intrinsic to the basic notion here, namely the use of the clusters in display (183).

17.2.4 Biclustering

An interesting and often useful variant of the clustering problem is one in which a doubly indexed set of observations x_{ij} for $i = 1, 2, \dots, I$ and $j = 1, 2, \dots, J$ (that might be thought of as laid out in an $I \times J$ two-way array or table) needs to be simultaneously be put into R (row) clusters over index i and C (columns) clusters over index j in such a way that the $R \times C$ cells are each homogeneous. Figure 42 portrays an $I = 6$ by $J = 12$ toy example with values of 72 univariate x_{ij} portrayed in "heat map" fashion. The object of simple biclustering is to regroup/rearrange rows and columns to make groups producing homogeneous "cells." We'll use the notation $r(i)$ for the row cluster index for data row i and $c(j)$ for the column cluster index for data column j .

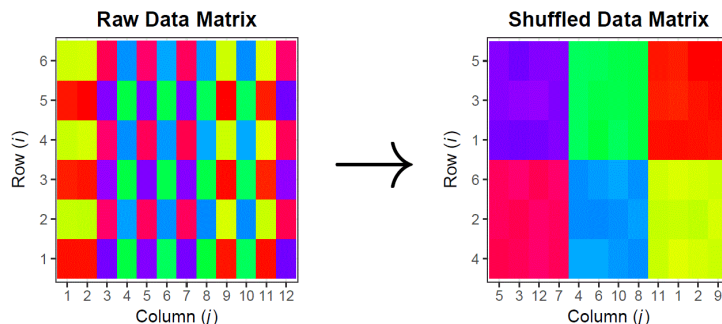


Figure 42: A toy 6×12 dataset clustered into $R = 2$ row clusters and $C = 3$ column clusters. (From Li, Reisner, Pham, Olafsson and Vardeman.) Values of x_{ij} s are portrayed in heat map fashion.

An Alternating Shuffling Algorithm An "alternating shuffling" algorithm of Li *et al.* for finding R good sets of rows and simultaneously C good sets of columns is based on a series of $R \times C$ matrices of means $\mathbf{M} = (m_{rc})$ and R row vectors of length J and C column vectors of length I with entries from rows and columns of \mathbf{M} .

1. One begins with some clustering of rows of the data matrix into R clusters and columns of the data matrix into C clusters, and computes for each (r, c) "cell" a sample mean of x_{ij} s with $r(i) = r$ and $c(j) = c$ (with row i in row cluster r and column j in column cluster c) creating an initial matrix \mathbf{M} .
2. For each $r = 1, 2, \dots, R$ one makes a new (J -dimensional) row vector "center" \mathbf{v}_r with j th entry $m_{rc(j)}$ assigned and re-clusters all rows in "K-means" fashion (assigning each row of values x_{ij} to the closest center using squared Euclidean \mathfrak{R}^J distance). With this new row clustering one recomputes the matrix of means \mathbf{M} .

3. For each $c = 1, 2, \dots, C$ one makes a new (I -dimensional) column vector "center" \mathbf{w}_c with i th entry $m_{r(i)c}$ and re-clusters all columns in "K-means" fashion (assigning each column of values x_{ij} to the closest center using squared Euclidean \Re^I distance). With this new column clustering one recomputes the matrix of means \mathbf{M} .
4. If $\sum_{i,j} (x_{ij} - m_{r(i)c(j)})^2$ is small and/or has ceased to decline with iterations, the algorithm terminates. Otherwise it returns to step 2.

Various "tweaks" are applied to this algorithm to deal with the eventuality that row or column clusters go empty. Multiple random starts are employed in the search of a good biclustering. The issue of what R and C should be used involves weighing complexity (large numbers of clusters) against a small value of the cell inhomogeneity criterion of step 4. All of this said, the algorithm is simple and effective, and appropriate modification of it allows the direct handling of even cases where not every cell of the $I \times J$ table is full.

Chakraborty's Bayes Biclustering The dissertation of Abhishek Chakraborty takes a Bayes modeling and analysis approach to biclustering univariate observations x_{ij} . To the notation above, add model parameters μ_{rc} for $r = 1, \dots, R$ and $c = 1, \dots, C$, and $\gamma^2 > 0$ and adopt a data model that given these parameters the $I \times J$ observations x_{ij} are independent with

$$x_{ij} \sim N(\mu_{r(i)c(j)}, \gamma^2)$$

Chakraborty's Bayes analysis then sets priors of independence for the vector $\mathbf{r} = (r(1), r(2), \dots, r(I))$, the vector $\mathbf{c} = (c(1), c(2), \dots, c(J))$, and the $R \times C$ means μ_{rc} .

A useful prior distribution for the means μ_{rc} is one of iid $N(0, \rho^2)$ variables for a parameter $\rho^2 > 0$. Useful priors for \mathbf{r} and \mathbf{c} are based on "Polya urn schemes." Take the case of \mathbf{r} . Let

$$n_q(\mathbf{r}) = \# [r(i) = q \text{ for } i = 1, 2, \dots, I]$$

and for an $\alpha > 0$ consider the distribution with pmf

$$g(\mathbf{r}) = \left(\prod_{i=1}^I \frac{1}{\alpha + i - 1} \right) \prod_{q \text{ s.t. } n_q(\mathbf{r}) > 0} \left(\left(\frac{\alpha}{I} \right) \left(\frac{\alpha}{I} + 1 \right) \cdots \left(\frac{\alpha}{I} + n_q(\mathbf{r}) - 1 \right) \right)$$

This symmetric distribution has the conditional distribution that

$$g(r(I) = r | r(1), r(2), \dots, r(I-1)) = \frac{\frac{\alpha}{I} + \# [r(i) = r \text{ for } i = 1, 2, \dots, I-1]}{\alpha + I - 1}$$

The case of a prior $h(\mathbf{c})$ is completely analogous. The parameters γ^2, ρ^2 , and α are treated as tuning parameters for the analysis.

This probability structure admits very simple Gibbs MCMC sampling and provides iterates from the posterior distribution over all of the means and (more

importantly) over the biclustering specified by the pair (\mathbf{r}, \mathbf{c}) . For a given pair (\mathbf{r}, \mathbf{c}) , rows i and i' with $r(i) = r(i')$ are clustered together, and columns j and j' with $c(j) = c(j')$ are clustered together. Observations x_{ij} and $x_{i'j'}$ with both $r(i) = r(i')$ and $c(j) = c(j')$ are in the same "cell" of the two-way clustering. The MCMC provides (through simple relative frequencies for iterates $(\mathbf{r}, \mathbf{c})^j$) approximate posterior probabilities that each pair of rows, each pair of columns, and each pair observations belong together in a clustering.

There are various ways to make use of the iterates representing the posterior distribution. One is to carry along with MCMC iterates $(\mathbf{r}, \mathbf{c})^j$ iterates of the means matrix \mathbf{M} (from the Li *et al.* algorithm) and identify an iterate with minimum $\sum_{i,j} (x_{ij} - m_{r(i)c(j)})^2$, using that iterate to represent the posterior distribution. Another (preferable) option is to identify a "central" iterate as follows. For two pairs (\mathbf{r}, \mathbf{c}) and $(\mathbf{r}^*, \mathbf{c}^*)$ one measure of their total disagreement in clustering of the x_{ij} s is

$$\begin{aligned} L((\mathbf{r}, \mathbf{c}), (\mathbf{r}^*, \mathbf{c}^*)) &= \sum_{(i,j),(i',j')} I[r(i) = r(i') \text{ and } c(j) = c(j')] I[r^*(i) \neq r^*(i') \text{ or } c^*(j) \neq c^*(j')] \\ &+ \sum_{(i,j),(i',j')} I[r(i) \neq r(i') \text{ or } r(j) \neq c(j')] I[r^*(i) = r^*(i') \text{ and } c^*(j) = c^*(j')] \end{aligned}$$

the total number of x_{ij} clustered together by only one of the two associated biclusterings. For fixed (\mathbf{r}, \mathbf{c}) one might take

$$\mathbf{L}((\mathbf{r}, \mathbf{c})) = \overline{L((\mathbf{r}, \mathbf{c}), (\mathbf{r}, \mathbf{c})^j)}$$

to be the arithmetic average across MCMC iterates of disagreement between clusterings of the x_{ij} s prescribed by (\mathbf{r}, \mathbf{c}) and by the iterates. An (\mathbf{r}, \mathbf{c}) minimizing this is a kind of central biclustering for representing the posterior, and while exact optimization of $\mathbf{L}((\mathbf{r}, \mathbf{c}))$ is computationally too hard, simply picking an iterate with smallest $\mathbf{L}((\mathbf{r}, \mathbf{c})^j)$ seems to be an effective way to represent the posterior and provide a single practically useful biclustering.

It is worth pointing out several things about this methodology. First, α is a kind of "prior sample size" and controls the distribution of the number of non-empty row (and column) clusters. Small α goes with posterior distributions for \mathbf{r} (or \mathbf{c}) concentrated on possible values with relatively few implied row (or column) clusters. Large α amounts to a prior for \mathbf{r} (or \mathbf{c}) with iid uniform coordinates, typically giving large weight to \mathbf{r} (or \mathbf{c}) values with many implied row (or column) clusters. Second, in this development, it is quite natural and effective to use values of R and C that are only loose upper bounds for seemingly appropriate numbers of row and column clusters, and let the analysis more or less sort out what numbers are genuinely plausible (in terms of the posterior distributions of non-zero $n_q(\mathbf{r})$ and $n_q(\mathbf{c})$). Finally, it is possible to allow some x_{ij} s to be unobserved in this development. In "missing at random" contexts, unobserved x_{ij} s can simply be treated as latent or auxiliary in the MCMC. And

for other contexts, modeling of censoring mechanisms provides Bayes analyses where missingness is informative about the value of an unobserved x_{ij} .

17.2.5 Self-Organizing Maps

For items $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_r$ belonging to \mathfrak{R}^p , the object here is to find $L \times M$ cluster centers/prototypes that adequately represent the items, where one wishes to think of those cluster centers/prototypes as indexed on an $L \times M$ regular grid in 2 dimensions (that one might take to be $\{1, 2, \dots, L\} \times \{1, 2, \dots, M\}$) with cluster centers/prototypes whose index vectors are close on the grid being close in \mathfrak{R}^p . (There could, of course, be 3-dimensional versions of this, and so on.) The object is both production of the set of centers/prototypes and assignment of data points to centers/prototypes. It thus amounts to some kind of modified/constrained $K = L \times M$ group clustering problem and simultaneous discovery of low-dimensional (typically 2-dimensional) structure in the items. This is illustrated in cartoon fashion in Figure 43.

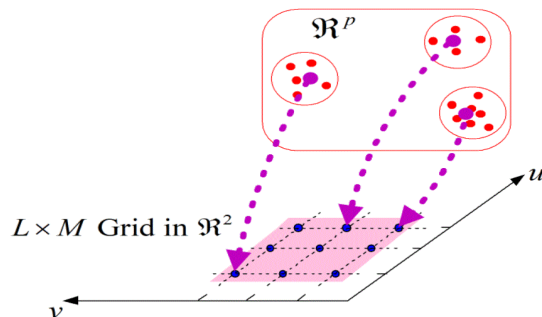


Figure 43: Cartoon of a Self-Organizing-Map assignment of points \mathbf{x} in \mathfrak{R}^p to cluster centers, themselves mapped to points on a grid in \mathfrak{R}^2 .

One will typically begin with standardization of the p coordinate variables x_j (so that $\sum \mathbf{x}_i = \mathbf{0}$ and the sample variance of each set of values $\{x_{ij}\}_{i=1,2,\dots,r}$ is 1). This puts all of the x_j on the same scale and doesn't allow one coordinate of an \mathbf{x}_i to dominate a Euclidean norm. Standard treatment of this topic seems to be driven by two somewhat ad hoc algorithms of Kohonen. Here we'll first describe those algorithms and then discuss a Bayes approach to the problem due to Zhou.

Kohonen's Algorithms One begins with some set of initial cluster centers $\{z_{lm}^0\}_{l=1,\dots,L \text{ and } m=1,\dots,M}$. This might be a random selection (without replacement or the possibility of duplication) from the set of items. It might be a set of grid points in the 2-dimensional plane in \mathfrak{R}^p defined by the first two principal components of the items $\{\mathbf{x}_i\}_{i=1,\dots,r}$. And there are surely other sensible

possibilities. Then define neighborhoods on the $L \times M$ grid, $\mathcal{N}(l, m)$, that are subsets of the grid "close" in some kind of distance (like regular Euclidean distance) to the various elements of the $L \times M$ grid. $\mathcal{N}(l, m)$ could be all of the grid, (l, m) alone, all grid points (l', m') within some constant 2-dimensional Euclidean distance of (l, m) , etc. Then define a weighting function on \mathfrak{R}^p , say $w(\|\mathbf{x}\|)$, so that $w(0) = 1$ and $w(\|\mathbf{x}\|) \geq 0$ is monotone non-increasing in $\|\mathbf{x}\|$. For some schedule of non-increasing positive constants $1 > \alpha_1 \geq \alpha_2 \geq \alpha_3 \geq \dots$, the SOM algorithms define iteratively sets of cluster centers/prototypes $\{\mathbf{z}_{lm}^j\}$ for $j = 1, 2, \dots$

At iteration j , an "online" version of SOM selects (randomly or perhaps in turn from an initially randomly set ordering of the items) an item \mathbf{x}^j and

1. identifies the center/prototype \mathbf{z}_{lm}^{j-1} closest to \mathbf{x}^j in to \mathfrak{R}^p , call it \mathbf{b}^j with corresponding grid coordinates $(l, m)^j$ (Izenman calls \mathbf{b}^j the "BMU" or best-matching-unit),
2. adjusts those \mathbf{z}_{lm}^{j-1} with index vectors belonging to $\mathcal{N}\left((l, m)^j\right)$ (close to the BMU index vector on the 2-dimensional grid) toward \mathbf{x}^j by the prescription

$$\mathbf{z}_{lm}^j = \mathbf{z}_{lm}^{j-1} + \alpha_j w\left(\left\|\mathbf{z}_{lm}^{j-1} - \mathbf{b}^j\right\|\right) \left(\mathbf{x}^j - \mathbf{z}_{lm}^{j-1}\right)$$

(adjusting those centers different from the BMU potentially less dramatically than the BMU), and

3. for those \mathbf{z}_{lm}^{j-1} with index pairs (l, m) not belonging $\mathcal{N}\left((l, m)^j\right)$ sets

$$\mathbf{z}_{lm}^j = \mathbf{z}_{lm}^{j-1}$$

iterating to convergence.

At iteration j , a "batch" version of SOM updates *all* centers/prototypes $\{\mathbf{z}_{lm}^{j-1}\}$ to $\{\mathbf{z}_{lm}^j\}$ as follows. For each \mathbf{z}_{lm}^{j-1} , let \mathcal{X}_{lm}^{j-1} be the set of items for which the closest element of $\{\mathbf{z}_{lm}^{j-1}\}$ has index pair (l, m) . Then update \mathbf{z}_{lm}^{j-1} as some kind of (weighted) average of the elements of $\cup_{(l,m)' \in \mathcal{N}(l,m)} \mathcal{X}_{(l,m)'}^{j-1}$ (the set of \mathbf{x}_i closest to prototypes with labels that are 2-dimensional grid neighbors of (l, m)). A natural form of this is to set (with $\bar{\mathbf{x}}_{(l,m)}^{j-1}$ the obvious sample mean of the elements of \mathcal{X}_{lm}^{j-1})

$$\mathbf{z}_{lm}^j = \frac{\sum_{(l,m)' \in \mathcal{N}(l,m)} w\left(\left\|\mathbf{z}_{lm}^{j-1} - \mathbf{z}_{(l,m)'}^{j-1}\right\|\right) \bar{\mathbf{x}}_{(l,m)'}^{j-1}}{\sum_{(l,m)' \in \mathcal{N}(l,m)} w\left(\left\|\mathbf{z}_{lm}^{j-1} - \mathbf{z}_{(l,m)'}^{j-1}\right\|\right)}$$

It is fairly obvious that even if these algorithms converge, different starting sets $\{\mathbf{z}_{lm}^0\}$ will produce different limits (symmetries alone mean, for example, that the choices $\mathbf{z}_{lm}^0 = \mathbf{u}_{lm}$ and $\mathbf{z}_{lm}^0 = \mathbf{u}_{L-l, M-m}$ produce what might look

like different limits, but are really completely equivalent). Beyond this, what is provided by the 2-dimensional layout of indices of prototypes is not immediately obvious. It seems to be fairly common to compare an error sum of squares for a SOM to that of a $K = L \times M$ means clustering and to declare victory if the SOM sum is not much worse than the K -means value.

Zhou's Bayesian SOM Dissertation work of Rick Zhou takes a principled Bayesian modeling and decision-theoretic approach to the SOM objective. The following is an overview of his methodology.

To develop a useful and "generative" model for $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_r$ belonging to \mathfrak{R}^p , begin by defining p (one for each dimension of the data vectors) 0 mean Gaussian spatial processes

$$\zeta_1(u, v), \zeta_2(u, v), \dots, \zeta_p(u, v)$$

and set

$$\zeta(u, v) = \begin{pmatrix} \zeta_1(u, v) \\ \vdots \\ \zeta_p(u, v) \end{pmatrix}$$

$\zeta(u, v)$ then defines a continuous random map $\mathfrak{R}^2 \rightarrow \mathfrak{R}^p$. For $L \times M$ points $\boldsymbol{\rho} = (l, m)$ on an integer grid in \mathfrak{R}^2 take $\zeta(l, m)$ as the center of a data-generating mechanism in \mathfrak{R}^p . Then assume that $\mathbf{x}_1, \dots, \mathbf{x}_r$ are iid as follows. First, one of the $L \times M$ fixed points $\boldsymbol{\rho} = (l, m)$ on the grid of interest is chosen at random, and then conditioned on this choice

$$\mathbf{x} \sim \text{MVN}_p(\zeta(\boldsymbol{\rho}), \boldsymbol{\Sigma}_\rho)$$

Upon supplying suitable (values of or) prior distributions for the parameters of

the p Gaussian processes and priors for the covariance matrices $\boldsymbol{\Sigma}_{l,m}$, MCMC will for observable $\mathbf{x}_1, \dots, \mathbf{x}_r$ and corresponding latent $\boldsymbol{\rho}_1, \dots, \boldsymbol{\rho}_r$ produce samples from a posterior distribution over all of

$$\begin{aligned} &\boldsymbol{\rho}_1, \boldsymbol{\rho}_2, \dots, \boldsymbol{\rho}_r \\ &\zeta_j(\boldsymbol{\rho}) \text{ for all points } \boldsymbol{\rho} \text{ in the grid and } j = 1, 2, \dots, p \\ &\boldsymbol{\Sigma}_\rho \text{ for all points } \boldsymbol{\rho} \text{ in the grid} \end{aligned}$$

What are of most interest are the grid points for the r cases, $\boldsymbol{\rho}_1, \dots, \boldsymbol{\rho}_r$. Two cases \mathbf{x}_i and $\mathbf{x}_{i'}$ belong to the same cluster if $\boldsymbol{\rho}_i = \boldsymbol{\rho}_{i'}$. The MCMC provides relative frequencies that approximate posterior probabilities that case i and case i' belong together, $P[\boldsymbol{\rho}_i = \boldsymbol{\rho}_{i'}]$. That is, one obtains an estimate $\hat{\mathbf{C}}$ of the matrix

$$\mathbf{C}_{r \times r} = (P[\boldsymbol{\rho}_i = \boldsymbol{\rho}_{i'}])_{\substack{i=1,2,\dots,r \\ i'=1,2,\dots,r}}$$

through MCMC relative frequencies. What one is then led to seek as a final work product is an assignment of data points to grid points that

1. is consistent with \mathbf{C} , and
2. (at least locally) more or less preserves relative distances between clusters in \mathfrak{R}^p in terms of distances between corresponding grid points in \mathfrak{R}^2 .

For a potential assignment of data points to grid points $\boldsymbol{\alpha}$ (that maps $\{1, 2, \dots, r\}$ to the set of indices $\boldsymbol{\rho} = (i, j)$ in the grid) we consider two types of penalties, one for inconsistency with \mathbf{C} and another for failure to preserve distances. First consider disagreement with \mathbf{C} . A measure of disparity between partitions of $\{1, 2, \dots, r\}$ corresponding to $\boldsymbol{\rho}_1, \dots, \boldsymbol{\rho}_r$ and to $\boldsymbol{\alpha}_1, \dots, \boldsymbol{\alpha}_r$ is for $a > 0$ and $b > 0$

$$L((\boldsymbol{\rho}_1, \dots, \boldsymbol{\rho}_r), (\boldsymbol{\alpha}_1, \dots, \boldsymbol{\alpha}_r)) = \sum_{i < i'} a I[\boldsymbol{\rho}_i = \boldsymbol{\rho}_{i'} \text{ and } \boldsymbol{\alpha}_i \neq \boldsymbol{\alpha}_{i'}] + \sum_{i < i'} b I[\boldsymbol{\rho}_i \neq \boldsymbol{\rho}_{i'} \text{ and } \boldsymbol{\alpha}_i = \boldsymbol{\alpha}_{i'}]$$

The average of this with respect to the posterior distribution is

$$a \sum_{i < i'} c_{i,i'} - (a + b) \sum_{i < i'} I[\boldsymbol{\alpha}_i = \boldsymbol{\alpha}_{i'}] \left(c_{i,i'} - \frac{b}{a + b} \right)$$

so a plausible penalty for inconsistency with \mathbf{C} is

$$R_1((\boldsymbol{\alpha}_1, \dots, \boldsymbol{\alpha}_r), \mathbf{C}, \lambda) = \frac{1}{r(r-1)} \sum_{n < n'} I[\boldsymbol{\alpha}_i = \boldsymbol{\alpha}_{i'}] (\lambda - c_{i,i'})$$

In the penalty $R_1((\boldsymbol{\alpha}_1, \dots, \boldsymbol{\alpha}_r), \mathbf{C}, \lambda)$ the parameter $\lambda \in (0, 1)$ determines what kinds of partitions of $\{1, 2, \dots, r\}$ are most heavily penalized. Large λ tends to heavily penalize $(\boldsymbol{\alpha}_1, \dots, \boldsymbol{\alpha}_r)$ prescribing large clusters, and small λ tends to heavily penalize $(\boldsymbol{\alpha}_1, \dots, \boldsymbol{\alpha}_r)$ with small clusters.

Consider then penalizing failure to preserve distances. Define maximum distances

$$M_{\text{grid}} = \max_{\boldsymbol{\rho} \text{ and } \boldsymbol{\rho}' \text{ on the grid}} \|\boldsymbol{\rho} - \boldsymbol{\rho}'\| \quad \text{and} \\ M_{\text{data}} = \max_{i, i'} \|\mathbf{x}_i - \mathbf{x}_{i'}\|$$

And define for $r \in \{1, 2, \dots, K\}$ the sets \mathcal{N}_K consisting of those pairs i and i' such that at least one of the points \mathbf{x}_i and $\mathbf{x}_{i'}$ is in the K -nearest neighborhood of the other. Then, a "local multi-dimensional scaling" type penalty⁴⁵ to apply to a potential assignment of data points to grid points is

$$R_2((\boldsymbol{\alpha}_1, \dots, \boldsymbol{\alpha}_r), K, \tau) = \frac{1}{K^2} \left\{ \begin{array}{l} \sum_{\substack{i < i' \text{ s.t.} \\ (i, i') \in \mathcal{N}_K}} \left(\frac{\|\mathbf{x}_i - \mathbf{x}_{i'}\|}{M_{\text{data}}} - \frac{\|\boldsymbol{\alpha}_i - \boldsymbol{\alpha}_{i'}\|}{M_{\text{grid}}} \right)^2 \\ - \tau \sum_{\substack{i < i' \text{ s.t.} \\ (i, i') \notin \mathcal{N}_K}} \frac{\|\boldsymbol{\alpha}_i - \boldsymbol{\alpha}_{i'}\|}{M_{\text{grid}}} \end{array} \right\}$$

⁴⁵See Section 17.3.

for a $\tau > 0$. (The first term penalizes failure to preserve local relative distances and the second encourages separation of mappings to points on the grid that are not neighbors in the \mathcal{R}^p dataset.)

So, in looking for a map α that is consistent with the posterior distribution and preserves local relative distances, a risk/figure of merit is for $\lambda > 0$

$$R\left(\left(\alpha_1, \dots, \alpha_r\right), \hat{\mathbf{C}}, \lambda, K, \gamma, \tau\right) = R_1\left(\left(\alpha_1, \dots, \alpha_r\right), \hat{\mathbf{C}}, \lambda\right) + \gamma R_2\left(\left(\alpha_1, \dots, \alpha_r\right), K, \tau\right)$$

Exact optimization of $R\left(\left(\alpha_1, \dots, \alpha_r\right), \hat{\mathbf{C}}, \lambda, K, \gamma, \tau\right)$ by choice of $\left(\alpha_1, \dots, \alpha_r\right)$ is in general an NP-hard problem and is thus rarely possible. What *is* possible and seems to work remarkably well is to make a long MCMC run (making one's estimate $\hat{\mathbf{C}}$ reliable) and then look for an MCMC iterate $\left(\rho_1^j, \dots, \rho_r^j\right)$ with the best value of $R\left(\left(\rho_1^j, \dots, \rho_r^j\right), \hat{\mathbf{C}}, \lambda, K, \gamma, \tau\right)$. The dissertation of Zhou provides substantial examples of the effectiveness of this strategy. The Bayes model behind the MCMC simply tends to concentrate the posterior (and thus make iterates) in a manner consistent with the clustering and distance preservation goals of SOM.

The famous "Wines" dataset has $p = 13$ chemical characteristics of $r = 178$ wine samples from 3 different cultivars (59 (red) samples, 71 (blue) samples, and 48 (violet) of the three types indexed 1-59, 60-130, and 131-178 respectively). Figure 44 is a graphical (grey-scale) representation of $\hat{\mathbf{C}}$ and a corresponding best iterate $\left(\rho_1^j, \dots, \rho_r^j\right)$.

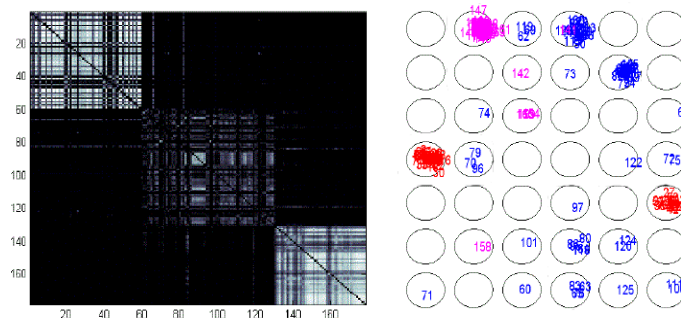


Figure 44: Bayes SOM representation of clustering of chemical characteristics vectors for $r = 178$ wine samples from 3 different cultivars. (From the PhD dissertation of Zhou.)

17.3 Multi-Dimensional Scaling

This material begins (as in Section 17.2) with dissimilarities among N items, d_{ij} , that might be collected in an $N \times N$ proximity matrix $\mathbf{D} = (d_{ij})$. (These

might, but do not necessarily, come from Euclidean distances among N data vectors $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ in \mathfrak{R}^p .) The object of multi-dimensional scaling is to (to the extent possible) represent the N items as points $\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_N$ in \mathfrak{R}^q with

$$\|\mathbf{z}_i - \mathbf{z}_j\| \approx d_{ij}$$

This is phrased precisely in terms of one of several optimization problems, where one seeks to minimize a "stress function" $S(\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_N)$.

The least squares (or Kruskal-Shepard) stress function (optimization criterion) is

$$S_{\text{LS}}(\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_N) = \sum_{i < j} (d_{ij} - \|\mathbf{z}_i - \mathbf{z}_j\|)^2$$

This criterion treats errors in reproducing big dissimilarities exactly like it treats errors in reproducing small ones. A different point of view would make faithfulness to small dissimilarities more important than the exact reproduction of big ones. The so-called Sammon mapping criterion

$$S_{\text{SM}}(\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_N) = \sum_{i < j} \frac{(d_{ij} - \|\mathbf{z}_i - \mathbf{z}_j\|)^2}{d_{ij}}$$

reflects this point of view.

Another approach to MDS that emphasizes the importance of small dissimilarities is discussed in HTF under the name of "local multi-dimensional scaling." Here one begins for fixed k with the symmetric set of index pairs

$$\mathcal{N}_k = \left\{ (i, j) \mid \begin{array}{l} \text{the number of } j' \text{ with } d_{ij'} < d_{ij} \text{ is less than } k \\ \text{or the number of } i' \text{ with } d_{i'j} < d_{ij} \text{ is less than } k \end{array} \right\}$$

(an index pair is in the set if one of the items is in the k -nearest neighbor neighborhood of the other). Then a stress function that emphasizes the matching of small dissimilarities and not large ones is (for some choice of $\tau > 0$)

$$S_{\text{L}}(\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_N) = \sum_{i < j \text{ and } (i, j) \in \mathcal{N}_k} (d_{ij} - \|\mathbf{z}_i - \mathbf{z}_j\|)^2 - \tau \sum_{i < j \text{ and } (i, j) \notin \mathcal{N}_k} \|\mathbf{z}_i - \mathbf{z}_j\|$$

Another version of MDS begins with similarities s_{ij} (rather than with dissimilarities d_{ij}). (One important special case of similarities derives from vectors $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ in \mathfrak{R}^p through centered inner products $s_{ij} \equiv \langle \mathbf{x}_i - \bar{\mathbf{x}}, \mathbf{x}_j - \bar{\mathbf{x}} \rangle$.) A "classical scaling" criterion is

$$S_{\text{C}}(\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_N) = \sum_{i < j} (s_{ij} - \langle \mathbf{z}_i - \bar{\mathbf{z}}, \mathbf{z}_j - \bar{\mathbf{z}} \rangle)^2$$

HTF claim that if in fact similarities are centered inner products, classical scaling is exactly equivalent to principal components analysis.

The four scaling criteria above are all "metric" scaling criteria in that the distances $\|\mathbf{z}_i - \mathbf{z}_j\|$ are meant to approximate the d_{ij} directly. An alternative is to attempt minimization of a non-metric stress function like

$$S_{\text{NM}}(\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_N) = \frac{\sum_{i < j} (\theta(d_{ij}) - \|\mathbf{z}_i - \mathbf{z}_j\|)^2}{\sum_{i < j} \|\mathbf{z}_i - \mathbf{z}_j\|^2}$$

over vectors $\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_N$ and increasing functions $\theta(\cdot)$. $\theta(\cdot)$ will preserve/enforce the natural ordering of dissimilarities without attaching importance to their precise values. Iterative algorithms for optimization of this stress function alternate between isotonic regression to choose $\theta(\cdot)$ and gradient descent to choose the \mathbf{z}_i .

In general, if one can produce a small value of stress in MDS, one has discovered a q -dimensional representation of N items, and for small q , this is a form of "simple structure."

17.4 More on Principal Components and Related Ideas

Here we extend the principal components ideas first raised in Section 2.4 based on the SVD ideas of Section 2.3, still with the motivation of using it as means for identifying simple structure in an N -case p -variable dataset, where as earlier, we write

$$\mathbf{X}_{N \times p} = \begin{pmatrix} \mathbf{x}'_1 \\ \mathbf{x}'_2 \\ \vdots \\ \mathbf{x}'_N \end{pmatrix}$$

17.4.1 "Sparse" Principal Components

In standard principal components analysis, the \mathbf{v}_j are sometimes called "loadings" because (in light of the fact that $\mathbf{z}_j = \mathbf{X}\mathbf{v}_j$) they specify what linear combinations of variables x_j are used in making the various principal component vectors. If the \mathbf{v}_j were "sparse" (had lots of 0s in them) interpretation of these loadings would be easier. So people have made proposals of alternative methods of defining "principal components" that will tend to produce sparse results. One due to Zou is as follows.

One might call a $\mathbf{v} \in \mathfrak{R}^p$ a first sparse principal component "direction"⁴⁶ if it is part of a minimizer (over choices of $\mathbf{v} \in \mathfrak{R}^p$ and $\boldsymbol{\theta} \in \mathfrak{R}^p$ with $\|\boldsymbol{\theta}\| = 1$) of the criterion

$$\sum_{i=1}^N \|\mathbf{x}_i - \boldsymbol{\theta}\mathbf{x}'_i\mathbf{v}\|^2 + \lambda \|\mathbf{v}\|^2 + \lambda_1 \|\mathbf{v}\|_1 \quad (184)$$

for $\|\cdot\|_1$ the l_1 norm on \mathfrak{R}^p and constants $\lambda \geq 0$ and $\lambda_1 \geq 0$. The last term in this expression is analogous to the lasso penalty on a vector of regression coefficients

⁴⁶We put quotes on "direction" because in this formulation \mathbf{v} will typically not be a unit vector.

as considered in Section 3.1.2, and produces the same kind of tendency to "0 out" entries that we saw in that context. If $\lambda_1 = 0$, \mathbf{v} is proportional to the ordinary first principal component direction. In fact, if $\lambda = \lambda_1 = 0$ and $N > p$, $\mathbf{v} = \boldsymbol{\theta}$ the ordinary first principal component direction *is* the optimizer.

For multiple components, an analogue of the first case is a set of K vectors $\mathbf{v}_k \in \Re^p$ organized into a $p \times K$ matrix \mathbf{V} that is part of a minimizer (over choices of $p \times K$ matrices \mathbf{V} and $p \times K$ matrices $\boldsymbol{\Theta}$ with $\boldsymbol{\Theta}'\boldsymbol{\Theta} = \mathbf{I}$) of the criterion

$$\sum_{i=1}^N \|\mathbf{x}_i - \boldsymbol{\Theta}\mathbf{V}'\mathbf{x}_i\|^2 + \lambda \sum_{k=1}^K \|\mathbf{v}_k\|^2 + \sum_{k=1}^K \lambda_{1k} \|\mathbf{v}_k\|_1 \quad (185)$$

for constants $\lambda \geq 0$ and $\lambda_{1k} \geq 0$. Zou has apparently provided effective algorithms for optimizing criteria (184) or (185).

17.4.2 Non-negative Matrix Factorization

There are contexts (for example, when data are counts) where it may not make intuitive sense to center inherently non-negative variables, so that \mathbf{X} is naturally non-negative, and one might want to find non-negative matrices \mathbf{W} and \mathbf{H} such that

$$\underset{N \times p}{\mathbf{X}} \approx \underset{N \times r}{\mathbf{W}} \underset{r \times p}{\mathbf{H}}$$

Here the emphasis might be on the columns of \mathbf{W} as representing "positive components" of the (positive) \mathbf{X} , just as the columns of the matrix \mathbf{UD} in SVDs provide the principal components of \mathbf{X} . Various optimization criteria could be set to guide the choice of \mathbf{W} and \mathbf{H} . One might try to minimize

$$\sum_{i=1}^N \sum_{j=1}^p \left(x_{ij} - (\mathbf{WH})_{ij} \right)^2$$

or maximize

$$\sum_{i=1}^N \sum_{j=1}^p \left(x_{ij} \ln (\mathbf{WH})_{ij} - (\mathbf{WH})_{ij} \right)$$

over non-negative choices of \mathbf{W} and \mathbf{H} , and various algorithms for doing these have been proposed. (Notice that the second of these criteria is an extension of a loglikelihood for independent Poisson variables with means entries in \mathbf{WH} to cases where the x_{ij} need only be non-negative, not necessarily integer.)

While at first blush this enterprise seems sensible, there is a lack of uniqueness in a factorization producing a product \mathbf{WH} , and therefore how to interpret the columns of one of the many possible \mathbf{W} s is not clear. (An easy way to see the lack of uniqueness is this. Suppose that all entries of the product \mathbf{WH} are positive. Then for \mathbf{E} a small enough (but not $\mathbf{0}$) matrix, all entries of $\mathbf{W}^* \equiv \mathbf{W}(\mathbf{I} + \mathbf{E}) \neq \mathbf{W}$ and $\mathbf{H}^* \equiv (\mathbf{I} + \mathbf{E})^{-1}\mathbf{H} \neq \mathbf{H}$ are positive, and $\mathbf{W}^*\mathbf{H}^* = \mathbf{WH}$.) Lacking some natural further restriction on the factors \mathbf{W} and \mathbf{H} (beyond non-negativity) it seems the practical usefulness of this basic idea is also lacking.

17.4.3 Archetypal Analysis

Another approach to finding an interpretable factorization of \mathbf{X} was provided by Cutler and Breiman in their "archetypal analysis." Again one means to write

$$\underset{N \times p}{\mathbf{X}} \approx \underset{N \times r}{\mathbf{W}} \underset{r \times p}{\mathbf{H}}$$

for appropriate \mathbf{W} and \mathbf{H} . But here two restrictions are imposed, namely that

1. the rows of \mathbf{W} are probability vectors (so that the approximation to \mathbf{X} is in terms of convex combinations/weighted averages of the rows of \mathbf{H}), and
2. $\underset{r \times p}{\mathbf{H}} = \underset{r \times N}{\mathbf{B}} \underset{N \times p}{\mathbf{X}}$ where the rows of \mathbf{B} are probability vectors (so that the rows of \mathbf{H} are in turn convex combinations/weighted averages of the rows of \mathbf{X}).

The r rows of $\mathbf{H} = \mathbf{B}\mathbf{X}$ are the "prototypes" (?archetypes?) used to represent the data matrix \mathbf{X} .

With this notation and restrictions, (stochastic matrices) \mathbf{W} and \mathbf{B} are chosen to minimize

$$\|\mathbf{X} - \mathbf{W}\mathbf{B}\mathbf{X}\|^2$$

It's clearly possible to rearrange the rows of a minimizing \mathbf{B} and make corresponding changes in \mathbf{W} without changing $\|\mathbf{X} - \mathbf{W}\mathbf{B}\mathbf{X}\|^2$. So strictly speaking, the optimization problem has multiple solutions. But in terms of the *set of rows of \mathbf{H}* (a set of prototypes of size r) it's possible that this optimization problem often has a unique solution. (Symmetries induced in the set of N rows of \mathbf{X} can be used to produce examples where it's clear that genuinely different sets of prototypes produce the same minimal value of $\|\mathbf{X} - \mathbf{W}\mathbf{B}\mathbf{X}\|^2$. But it seems likely that real datasets will usually lack such symmetries and lead to a single optimizing set of prototypes.)

Emphasis in this version of the "approximate \mathbf{X} " problem is on the set of prototypes as "representative data cases." This has to be taken with a grain of salt, since they are nearly always near the "edges" of the dataset. This should be no surprise, as line segments between extreme cases in \mathfrak{R}^p can be made to run close to cases in the "middle" of the dataset, while line segments between interior cases in the dataset will never be made to run close to extreme cases.

17.4.4 Independent Component Analysis

We begin by supposing (as before) that \mathbf{X} has been centered. For simplicity, suppose also that it is full rank (rank p). With the SVD as before

$$\underset{N \times p}{\mathbf{X}} = \underset{N \times p}{\mathbf{U}} \underset{p \times p}{\mathbf{D}} \underset{p \times p}{\mathbf{V}'}$$

we consider the "sphered" version of the data matrix

$$\mathbf{X}^* = \sqrt{N} \mathbf{X} \mathbf{V} \mathbf{D}^{-1}$$

so that the sample covariance matrix of the data is

$$\frac{1}{N} (\mathbf{X}^{*'} \mathbf{X}^*) = \mathbf{I}$$

Note that the columns of \mathbf{X}^* are then scaled principal components of the (centered) data matrix and we operate with and on \mathbf{X}^* . (For simplicity of notation, we'll henceforth drop the "*" on \mathbf{X} .) This methodology seems to be an attempt to find latent probabilistic structure in terms of independent variables to account for the principal components.

In particular, in its linear form, ICA attempts to model the N (transposed) rows of \mathbf{X} as iid of the form

$$\underset{p \times 1}{\mathbf{x}_i} = \underset{p \times p}{\mathbf{A}} \underset{p \times 1}{\mathbf{s}_i} \quad (186)$$

for iid vectors \mathbf{s}_i , where the (marginal) distribution of the vectors \mathbf{s}_i is one of independence of the p coordinates/components and the matrix \mathbf{A} is an unknown parameter. Consistent with our sphering of the data matrix, we'll assume that $\text{Cov} \mathbf{x} = \mathbf{I}$ and without any loss of generality assume that the covariance matrix for \mathbf{s} is not only diagonal, but that $\text{Cov} \mathbf{s} = \mathbf{I}$. Since then $\mathbf{I} = \text{Cov} \mathbf{x} = \mathbf{A} (\text{Cov} \mathbf{s}) \mathbf{A}' = \mathbf{A} \mathbf{A}'$, \mathbf{A} must be orthogonal, and so

$$\mathbf{A}' \mathbf{x} = \mathbf{s}$$

Obviously, if one can estimate \mathbf{A} with an orthogonal $\hat{\mathbf{A}}$ then $\hat{\mathbf{s}}_i \equiv \hat{\mathbf{A}}' \mathbf{x}_i$ serves as an estimate of what vector of independent components led to the i th row of \mathbf{X} and indeed

$$\hat{\mathbf{S}} \equiv \mathbf{X} \hat{\mathbf{A}}$$

has columns that provide predictions of the N (row) p -vectors \mathbf{s}'_i , and we might thus call those the "independent components" of \mathbf{X} (just as we term the columns of $\mathbf{X} \mathbf{V}$ the principal components of \mathbf{X}). There is a bit of arbitrariness in the representation (186) because the ordering of the coordinates of \mathbf{s} and the corresponding rows of \mathbf{A} is arbitrary. But this is no serious concern.

So then, the question is what one might use as a method to estimate \mathbf{A} in display (186). There are several possibilities. The one discussed in HTF is related to entropy and Kullback-Leibler distance. If one assumes that a (m -dimensional) random vector \mathbf{Y} has a density p with marginal densities p_1, p_2, \dots, p_m then an "independence version" of the distribution of \mathbf{Y} has density $\prod_{j=1}^m p_j$ and the (non-negative) K-L divergence of the distribution of \mathbf{Y} from

its independence version is

$$\begin{aligned}
KL\left(p, \prod_{j=1}^m p_j\right) &= \int p(\mathbf{y}) \ln\left(\frac{p(\mathbf{y})}{\prod_{j=1}^m p_j(y_j)}\right) d\mathbf{y} \\
&= \int p(\mathbf{y}) \ln p(\mathbf{y}) d\mathbf{y} - \sum_{j=1}^m \int p(\mathbf{y}) (\ln p_j(y_j)) d\mathbf{y} \\
&= \int p(\mathbf{y}) \ln p(\mathbf{y}) d\mathbf{y} - \sum_{j=1}^m \int p_j(y_j) (\ln p_j(y_j)) dy_j \\
&= \sum_{j=1}^m \mathcal{H}(Y_j) - \mathcal{H}(\mathbf{Y})
\end{aligned}$$

for \mathcal{H} the entropy function for a random argument. Since entropy is an inverse measure of information for a distribution, this K-L divergence is a difference in the information carried by \mathbf{Y} (jointly) and the sum across the components of their individual information contents. If it is small, one might loosely interpret the components of \mathbf{Y} as approximately independent.

If one then thinks of \mathbf{s} as random and of the form $\mathbf{A}'\mathbf{x}$ for random \mathbf{x} , it is perhaps sensible to seek an orthogonal \mathbf{A} to minimize (for for \mathbf{a}_j the j th column of \mathbf{A})

$$\begin{aligned}
\sum_{j=1}^p \mathcal{H}(s_j) - \mathcal{H}(\mathbf{s}) &= \sum_{j=1}^p \mathcal{H}(\mathbf{a}'_j \mathbf{x}) - \mathcal{H}(\mathbf{A}'\mathbf{x}) \\
&= \sum_{j=1}^p \mathcal{H}(\mathbf{a}'_j \mathbf{x}) - \mathcal{H}(\mathbf{x}) - \ln |\det \mathbf{A}| \\
&= \sum_{j=1}^p \mathcal{H}(\mathbf{a}'_j \mathbf{x}) - \mathcal{H}(\mathbf{x})
\end{aligned}$$

As it turns out, this is equivalent (for orthogonal \mathbf{A}) to maximization of

$$C(\mathbf{A}) = \sum_{j=1}^p (\mathcal{H}(z) - \mathcal{H}(\mathbf{a}'_j \mathbf{x})) \quad (187)$$

for z standard normal. Then a common approximation is apparently

$$(\mathcal{H}(z) - \mathcal{H}(\mathbf{a}'_j \mathbf{x})) \approx (\text{EG}(z) - \text{EG}(\mathbf{a}'_j \mathbf{x}))^2$$

for $G(u) \equiv \frac{1}{c} \ln \cosh(cu)$ for a $c \in [1, 2]$. Then, criterion (187) has the empirical approximation

$$\widehat{C}(\mathbf{A}) = \sum_{j=1}^p \left(\text{EG}(z) - \frac{1}{N} \sum_{i=1}^N G(\mathbf{a}'_j \mathbf{x}'_i) \right)^2$$

where, as usual, \mathbf{x}'_i is the i th row of \mathbf{X} . $\widehat{\mathbf{A}}$ can be taken to be an optimizer of $\widehat{C}(\mathbf{A})$.

Ultimately, this development produces a rotation matrix that makes the p entries of rotated and scaled principal component score vectors "look as independent as possible." This is thought of as resolution of a data matrix into its "independent sources" and as a technique for "blind source separation."

17.4.5 Principal Curves and Surfaces

In the context of Section 2.4, the line in \mathfrak{R}^p defined by $\{c\mathbf{v}_1 | c \in \mathfrak{R}\}$ serves as a best straight line representative of the dataset. Similarly, the "plane" in \mathfrak{R}^p defined by $\{c_1\mathbf{v}_1 + c_2\mathbf{v}_2 | c_1 \in \mathfrak{R}, c_2 \in \mathfrak{R}\}$ serves as a best "planar" representative of the dataset. The notions of "principal curve" and "principal surface" are attempts to generalize these ideas to 1-dimensional and 2-dimensional structures in \mathfrak{R}^p that may not be "straight" or "flat" but rather have some curvature.

A parametric curve in \mathfrak{R}^p is represented by a vector-valued function

$$\mathbf{h}(t) = \begin{pmatrix} h_1(t) \\ h_2(t) \\ \vdots \\ h_p(t) \end{pmatrix}$$

defined on some interval $[0, T]$, where we assume that the coordinate functions $h_j(t)$ are smooth. With

$$\mathbf{h}'(t) = \begin{pmatrix} h'_1(t) \\ h'_2(t) \\ \vdots \\ h'_p(t) \end{pmatrix}$$

the "velocity vector" for the curve, $\|\mathbf{h}'(t)\|$ is then the "speed" for the curve and the arc length (distance) along $\mathbf{h}(t)$ from $t = 0$ to $t = t'$ is

$$L_{\mathbf{h}}(t') = \int_0^{t'} \|\mathbf{h}'(t)\| dt$$

In order to set an unambiguous representation of a curve, it will be useful to assume that it is parameterized so that it has unit speed, i.e. that $L_{\mathbf{h}}(t') = t'$ for all $t' \in [0, T]$. Notice that if it does not, for $(L_{\mathbf{h}})^{-1}$ the inverse of the arc length function, the parametric curve

$$\mathbf{g}(\lambda) = \mathbf{h}\left((L_{\mathbf{h}})^{-1}(\lambda)\right) \quad \text{for } \lambda \in [0, L_{\mathbf{h}}(T)] \quad (188)$$

does have unit speed and traces out the same set of points in \mathfrak{R}^p that are traced out by $\mathbf{h}(t)$. So there is no loss of generality in assuming that parametric curves we consider here are parameterized by arc length, and we'll henceforth write $\mathbf{h}(\lambda)$.

Then, for a unit speed parametric curve $\mathbf{h}(\lambda)$ and point $\mathbf{x} \in \mathfrak{R}^p$, we'll define the projection index

$$\lambda_{\mathbf{h}}(\mathbf{x}) = \sup_{\lambda} \left\{ \lambda \mid \|\mathbf{x} - \mathbf{h}(\lambda)\| = \inf_{\lambda} \|\mathbf{x} - \mathbf{h}(\lambda)\| \right\} \quad (189)$$

This is roughly the last arc length for which the distance from \mathbf{x} to the curve is minimum. If one thinks of \mathbf{x} as random, the "reconstruction error"

$$E \|\mathbf{x} - \mathbf{h}(\lambda_{\mathbf{h}}(\mathbf{x}))\|^2$$

(the expected squared distance between \mathbf{x} and the curve) might be thought of as a measure of how well the curve represents the distribution. Of course, for a dataset containing N cases \mathbf{x}_i , an empirical analog of this is

$$\frac{1}{N} \sum_{i=1}^N \|\mathbf{x}_i - \mathbf{h}(\lambda_{\mathbf{h}}(\mathbf{x}_i))\|^2 \quad (190)$$

and a "good" curve representing the dataset should have a small value of this empirical reconstruction error. Notice however, that this can't be the only consideration. If it was, there would sure be no real difficulty in running a very wiggly (and perhaps very long) curve through every element of a dataset to produce a curve with 0 empirical reconstruction error. This suggests that with

$$\mathbf{h}''(\lambda) = \begin{pmatrix} h_1''(\lambda) \\ h_2''(\lambda) \\ \vdots \\ h_p''(\lambda) \end{pmatrix}$$

the curve's "acceleration vector," there must be some kind of control exercised on the curvature, $\|\mathbf{h}''(\lambda)\|$, in the search for a good curve. We'll note below where this control is implicitly applied in standard algorithms for producing principal curves for a dataset.

Returning for a moment to the case where we think of \mathbf{x} as random, we'll say that $\mathbf{h}(\lambda)$ is a principal curve for the distribution of \mathbf{x} if it satisfies a so-called self-consistency property, namely that

$$\mathbf{h}(\lambda) = E[\mathbf{x} \mid \lambda_{\mathbf{h}}(\mathbf{x}) = \lambda] \quad (191)$$

This provides motivation for an iterative 2-step algorithm to produce a "principal curve" for a dataset.

Begin iteration with $\mathbf{h}^0(\lambda)$ the ordinary first principal component line for the dataset. Specifically, do something like the following. For some choice $T > 2 \max_{i=1, \dots, N} |\langle \mathbf{x}_i, \mathbf{v}_1 \rangle|$ begin with

$$\mathbf{h}^0(\lambda) = \left(\lambda - \frac{T}{2} \right) \mathbf{v}_1$$

to create a unit-speed curve that extends past the dataset in both directions along the first principal component direction in \mathfrak{R}^p . Then project the \mathbf{x}_i onto the line to get N values

$$\lambda_i^1 = \lambda_{\mathbf{h}^0}(\mathbf{x}_i) = \langle \mathbf{x}_i, \mathbf{v}_1 \rangle + \frac{T}{2}$$

and in light of the criterion (191) more or less average \mathbf{x}_i with corresponding $\lambda_{\mathbf{h}^0}(\mathbf{x}_i)$ near λ to get $\mathbf{h}^1(\lambda)$. A specific possible version of this is to consider, for each coordinate, j , the N pairs

$$\{(\lambda_i^1, x_{ij})\}$$

and to take as $h_j^1(\lambda)$ a function on $[0, T]$ that is a 1-dimensional cubic smoothing spline. One may then assemble these into a vector function to create $\mathbf{h}^1(\lambda)$. NOTICE that implicit in this prescription is control over the second derivatives of the component functions through the stiffness parameter/weight in the smoothing spline optimization. Notice also that for this prescription, the unit-speed property of $\mathbf{h}^0(\lambda)$ will not carry over to $\mathbf{h}^1(\lambda)$ and it seems that one must use the idea (188) to assure that $\mathbf{h}^1(\lambda)$ is parameterized in terms of arc length on $[0, \int_0^T \|\mathbf{h}^1(\lambda)\| d\lambda]$.

With iterate $\mathbf{h}^{m-1}(\lambda)$ in hand, one projects the \mathbf{x}_i onto the curve to get N values

$$\lambda_i^m = \lambda_{\mathbf{h}^{m-1}}(\mathbf{x}_i)$$

and for each coordinate j considers the N pairs

$$\{(\lambda_i^m, x_{ij})\}$$

Fitting a 1-dimensional cubic smoothing spline to these produces $h_j^m(\lambda)$, and these functions are assembled into a vector to create $\mathbf{h}^m(\lambda)$ (which may need some adjustment via relationship (188) to assure that the curve is parameterized in terms of arc length). One iterates until the empirical reconstruction error (190) converges, and one takes the corresponding $\mathbf{h}^m(\lambda)$ to be a principal curve. Notice that which stiffness parameter is applied in the smoothing steps will govern what one gets for such a curve.

There have been efforts to extend principal curves technology to the creation of 2-dimensional principal surfaces. Parts of the extension are more or less clear. A parametric surface in \mathfrak{R}^p is represented by a vector-valued function

$$\mathbf{h}(\mathbf{t}) = \begin{pmatrix} h_1(\mathbf{t}) \\ h_2(\mathbf{t}) \\ \vdots \\ h_p(\mathbf{t}) \end{pmatrix}$$

for $\mathbf{t} \in S \subset \mathfrak{R}^2$. A projection index parallel to form (189) and self-consistency property parallel to that in display (191) can clearly be defined for the surface

case, and thin plate splines can replace 1-dimensional cubic smoothing splines for producing iterates of coordinate functions. But ideas of unit-speed don't have obvious translations to \mathfrak{R}^2 and methods here seem fundamentally more complicated than what is required for the 1-dimensional case.

17.5 (Original) Google PageRanks

This might be thought of as of some general interest beyond the particular application of ranking Web pages, if one abstracts the general notion of summarizing features of a directed graph with N nodes (N Web pages in the motivating application) where edges point from some nodes to other nodes (there are links on some Web pages to other Web pages). The basic idea is that one wishes to rank the nodes (Web pages) by some measure of importance.

If $i \neq j$ define

$$L_{ij} = \begin{cases} 1 & \text{if there is a directed edge pointing from node } j \text{ to node } i \\ 0 & \text{otherwise} \end{cases}$$

and define

$$c_j = \sum_{i=1}^N L_{ij} = \text{the number of directed edges pointed away from node } j$$

(There is the question of how we are going to define L_{jj} . We may either declare that there is an implicit edge pointed from each node j to itself and adopt the convention that $L_{jj} = 1$ or we may declare that all $L_{jj} = 0$.)

A node (Web page) might be more important if many other (particularly, important) nodes have edges (links) pointing to it. The Google PageRanks $r_i > 0$ are chosen to satisfy⁴⁷

$$r_i = (1 - d) + d \sum_j \left(\frac{L_{ij}}{c_j} \right) r_j \quad (192)$$

for some $d \in (0, 1)$ (producing minimum rank $(1 - d)$). (Apparently, a standard choice is $d = .85$.) The question is how one can identify the r_i .⁴⁸

Without loss of generality, with

$$\mathbf{r} = \begin{pmatrix} r_1 \\ \vdots \\ r_N \end{pmatrix}$$

⁴⁷There is the question of what $\frac{L_{ij}}{c_j}$ should mean in case $c_j = 0$. We'll presume that the meaning is that the ratio is 0.

⁴⁸These are, of course, simply N linear equations in the N unknowns r_i , and for small N one might ignore special structure and simply solve these numerically with a generic solver. In what follows we exploit some special structure.

we'll assume that $\mathbf{r}'\mathbf{1} = N$ so that the average rank is 1. Then, for

$$d_j = \begin{cases} \frac{1}{c_j} & \text{if } c_j \neq 0 \\ 0 & \text{if } c_j = 0 \end{cases}$$

define the $N \times N$ diagonal matrix $\mathbf{D} = \mathbf{diag}(d_1, d_2, \dots, d_N)$. (Clearly, if we use the $L_{jj} = 1$ convention, then $d_j = 1/c_j$ for all j .) Then in matrix form, the N equations (192) are (for $\mathbf{L} = (L_{ij})_{\substack{i=1,\dots,N \\ j=1,\dots,N}}$)

$$\begin{aligned} \mathbf{r} &= (1 - d)\mathbf{1} + d\mathbf{L}\mathbf{D}\mathbf{r} \\ &= \left(\frac{1}{N}(1 - d)\mathbf{1}\mathbf{1}' + d\mathbf{L}\mathbf{D} \right) \mathbf{r} \end{aligned}$$

(using the assumption that $\mathbf{r}'\mathbf{1} = N$). Let

$$\mathbf{T} = \left(\frac{1}{N}(1 - d)\mathbf{1}\mathbf{1}' + d\mathbf{L}\mathbf{D} \right)$$

so that $\mathbf{r}'\mathbf{T}' = \mathbf{r}'$.

Note all entries of \mathbf{T} are non-negative and that

$$\begin{aligned} \mathbf{T}'\mathbf{1} &= \left(\frac{1}{N}(1 - d)\mathbf{1}\mathbf{1}' + d\mathbf{L}\mathbf{D} \right)' \mathbf{1} \\ &= \frac{1}{N}(1 - d)\mathbf{1}\mathbf{1}'\mathbf{1} + d\mathbf{D}\mathbf{L}'\mathbf{1} \\ &= (1 - d)\mathbf{1} + d\mathbf{D} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{pmatrix} \end{aligned}$$

so that if all $c_j > 0$, $\mathbf{T}'\mathbf{1} = \mathbf{1}$. We have this condition as long as we either limit application to sets of nodes (Web pages) where each node has an outgoing edge (an outgoing link) or we decide to count every node as pointing to itself (every page as linking to itself) using the $L_{jj} = 1$ convention. Henceforth suppose that indeed all $c_j > 0$.

Under this assumption, \mathbf{T}' is a stochastic matrix (with rows that are probability vectors), the transition matrix for an irreducible aperiodic finite state Markov Chain. Defining the probability vector

$$\mathbf{p} = \frac{1}{N}\mathbf{r}$$

it then follows that since $\mathbf{p}'\mathbf{T}' = \mathbf{p}'$ the PageRank vector is N times the stationary probability vector for the Markov Chain. This stationary probability vector can then be found as the limit of any row of

$$(\mathbf{T}')^n$$

as $n \rightarrow \infty$.

Part VI

Miscellanea

18 Graphs as Representing Independence Relationships in Multivariate Distributions

The most coherent approaches to statistical machine learning are ultimately based on probability models for the generation of all of (\mathbf{x}, y) and additionally "all" other "relevant" unobserved/latent variables. Even for small N , such multivariate distributions are in general impossibly complicated and impossible to detail on the basis of a training set. Only by making simplifying assumptions can progress be made. Graphs are often called upon to organize and represent useful simplifying assumptions about conditional independence between various of the variables to be jointly modeled, and their use is sometime treated as an important part of machine learning.

Random quantities X and Y are **conditionally independent given Z** written

$$X \perp\!\!\!\perp Y \mid Z$$

provided densities factor as

$$f_{X,Y|Z}(x, y|z) = f_{X|Z}(x|z) f_{Y|Z}(y|z)$$

A basic result about conditional independence is that

$$X \perp\!\!\!\perp Y \mid Z \iff f_{X|Y,Z}(x|y, z) = f_{X|Z}(x|z)$$

Conditional independence (like ordinary independence) has some important/useful properties/implications. Among these are

1. $X \perp\!\!\!\perp Y \mid Z \Rightarrow Y \perp\!\!\!\perp X \mid Z$,
2. $X \perp\!\!\!\perp Y \mid Z$ and $U = h(X) \Rightarrow U \perp\!\!\!\perp Y \mid Z$,
3. $X \perp\!\!\!\perp Y \mid Z$ and $U = h(X) \Rightarrow X \perp\!\!\!\perp Y \mid Z, U$,
4. $X \perp\!\!\!\perp Y \mid Z$ and $X \perp\!\!\!\perp W \mid (Y, Z) \Rightarrow X \perp\!\!\!\perp (W, Y) \mid Z$, and
5. $X \perp\!\!\!\perp Y \mid Z$ and $X \perp\!\!\!\perp Z \mid Y \Rightarrow X \perp\!\!\!\perp (Y, Z)$.

A possibly more natural (but equivalent) version of property 3. is

$$X \perp\!\!\!\perp Y \mid Z \text{ and } U = h(X) \Rightarrow Y \perp\!\!\!\perp (X, U) \mid Z$$

A main goal of this material is representing aspects of large joint distributions in ways that allow one to "see" conditional independence relationships in graphs representing them and to construct correspondingly simple joint and conditional densities for variables. In this section we will provide a brief introduction to the simplest ideas in this enterprise. More on the topics can be found in books by Murphy, by Wasserman, and by Lauritzen. We'll first consider what relationships are typically represented using directed graphs, and then what relationships are represented using undirected graphs.

18.1 Some Considerations for Directed Graphical Models

A **directed graph** (that might potentially represent some aspects of the joint distribution of (X, Y, Z, \dots)) consists of **nodes** (or vertices) X, Y, Z, \dots and **arrows** (or edges) pointing between some of them. A corresponding probability model for (X, Y, Z, \dots) can variously be known as a **directed graphical model**, a **Bayes network** (though there is nothing intrinsically Bayesian about this material), a **belief network** (though there need be nothing subjective about what it represents), or a **causal network** (though again, there is nothing inherently causal about what it represents).

For a graph with nodes/vertices X, Y, Z, \dots

1. if an arrow points from X to Y we will say that X is a **parent** of Y and that Y is a **child** of X ,
2. a sequence of arrows beginning at X and ending at Y will be called a **directed path** from X to Y ,
3. if $X = Y$ or there is a directed path from X to Y , we will say that X is an **ancestor** of Y and Y is a **descendent** of X ,
4. a directed path that starts and ends at the same vertex is called a **cycle**, and
5. a directed graph is **acyclic** if it has no cycles.

As a matter of notation/shorthand an acyclic directed graph is usually called a **DAG** (a directed acyclic graph) although the corresponding word order is not really as good as that corresponding to the unpronounceable acronym "ADG."

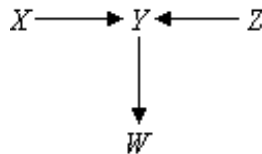


Figure 45: A DAG.

In Figure 45, X is a parent of Y and an ancestor of W . There is a directed path from X to W . Y is a child of both X and Z .

For a vector of random quantities (and vertices) $\mathbf{X} = (X_1, X_2, \dots, X_k)$ and a distribution P for \mathbf{X} , it is said that a DAG \mathcal{G} **represents** P (or P is **Markov to** \mathcal{G}) if and only if densities satisfy

$$p_{\mathbf{X}}(\mathbf{x}) = \prod_{i=1}^k p(x_i | \text{parents}_i) \quad (193)$$

where

$$\text{parents}_i = \{\text{parents of } X_i \text{ in the DAG } \mathcal{G}\}$$

So a joint distribution P for (X, Y, Z, W) is represented by the DAG pictured in Figure 45 if and only if

$$p_{X,Y,Z,W}(x, y, z, w) = p_X(x) p_Z(z) p_{Y|X,Z}(y|x, z) p_{W|Y}(w|y) \quad (194)$$

A condition equivalent to the Markov condition can be stated in terms of conditional independence relationships. That is, let \tilde{X}_i stand for the set of all vertices X_1, X_2, \dots, X_k in a DAG \mathcal{G} except for the parents and descendants of X_i . Then

$$P \text{ is represented by } \mathcal{G} \Leftrightarrow \text{for every vertex } X_i, X_i \perp\!\!\!\perp \tilde{X}_i | \text{parents}_i \quad (195)$$

So, for example, if a joint distribution P for (X, Y, Z, W) is represented by the DAG pictured in Figure 45 it follows that

$$X \perp\!\!\!\perp Z \text{ and } W \perp\!\!\!\perp (X, Z) | Y$$

Condition (195) provides a way to simply identify some conditional independence relationships implied by a DAG representation of a joint distribution P . Upon introducing some more concepts and machinery (concerning "connectedness" and "separatedness" of vertices of a DAG) other conditional independence relationships that will always hold for P represented by \mathcal{G} can be identified. We refer the interested reader to the the books of Murphy, Wasserman, and Lauritzen for more details. Rather than going further into the probabilistic implications of various types of structure possessed by a DAG \mathcal{G} representing a distribution P , we will here simply consider in broad terms the practical implications and difficulties associated with adopting a directed graphical model.

In the first place, a directed graphical model is less complicated than a general distribution for the same set of variables, and thus potentially requires less data to accurately characterize. As a simple toy example, consider a jointly discrete distribution for (X, Y, Z, W) taking values in a simple set $\{1, 2, 3\}^4$. A general joint distribution for (X, Y, Z, W) requires the specification of $3^4 - 1 = 80$ probabilities (the 81st coming from the fact that values $p_{X,Y,Z,W}(x, y, z, w)$ must sum to 1). On the other hand, if P is represented by the graph \mathcal{G} in Figure 45, then form (194) holds and only 2 values are needed to specify $p_X(x)$, 2 values

are needed to specify $p_Z(z)$, 2 values are needed to specify each of 9 different conditional pmfs $p_{Y|X,Z}(y|x,z)$, and finally 2 values are needed to specify each of 3 conditional pmfs $p_{W|Y}(w|y)$. That is, there are $2 + 2 + 2(9) + 2(3) = 28$ probabilities to be specified under form (194), far fewer than in general.

None of this touches the obvious questions of what forms of DAG are appropriate (and why they are so) in particular applications and lead to effective methods of translating a training set into appropriate estimates for the factors $p(x_i|parents_i)$ in the expression (193). The question of how to infer the factors of the product form from training data is particularly perplexing for models that include latent/hidden/unobserved nodes. Researchers who value this kind of modeling must obviously produce tractable and believable DAGs and corresponding forms for conditional distributions that lead to effective specialized fitting methods for the kinds of training data they expect to encounter.

18.2 Some Considerations for Undirected Graphical Models

An **undirected graph** (that might potentially represent some aspects of the joint distribution of $\mathbf{X} = (X_1, X_2, \dots, X_k)$) consists of **nodes** (or vertices) X_1, X_2, \dots, X_k and edges connecting some of them. A corresponding probability model for \mathbf{X} is variously known as an **undirected graphical model**, a **Markov random field**, and a **Markov network**.

Some of the terminology introduced for directed graphs carries over to undirected graphs. And there are also some important additional concepts. For a graph with nodes/vertices X, Y, Z, \dots

1. two vertices X and Y are said to be **adjacent** if there is an edge between them, here symbolized as $X \sim Y$,
2. a sequence of vertices $\{X_1, X_2, \dots, X_n\}$ is a **path** if $X_i \sim X_{i+1}$ for each i ,
3. if A, B , and C are disjoint sets of vertices, C **separates A and B** provided every path from a vertex $X \in A$ to a vertex $Y \in B$ contains an element of C ,
4. a **clique** is a set of vertices of a graph that are all adjacent to each other, and
5. a clique is **maximal** if it is not possible to add another vertex to it and still have a clique.

Item 3. could be equivalently stated as " C **separates A and B** provided upon removing the vertices in C from the graph there is no path from a vertex in A to vertex in B ."

The simple example below can be used to illustrate some of this terminology. In Figure 46 $\{X_1, X_3\}$ and $\{X_4\}$ are separated by $\{X_2\}$, $\{X_3\}$ and $\{X_4\}$ are separated by $\{X_2\}$, $\{X_1, X_2\}$ is a clique, and $\{X_1, X_2, X_3\}$ is a maximal clique.

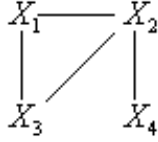


Figure 46: An undirected graph.

For a vector of random quantities (and vertices) $\mathbf{X} = (X_1, X_2, \dots, X_k)$ for each i and j let $\check{\mathbf{X}}_{ij}$ stand for all elements of $\{X_1, X_2, \dots, X_k\}$ except elements i and j . For P the distribution of \mathbf{X} , we may associate with P a **pairwise Markov graph** \mathcal{G} by

failing to connect X_i and X_j with an edge if and only if $X_i \perp\!\!\!\perp X_j \mid \check{\mathbf{X}}_{ij}$

A pairwise Markov graph for P can be made by considering only $\binom{k}{2}$ pairwise conditional independence questions. But as it turns out, many other conditional independence relationships can be read from it. That is, it turns out that if \mathcal{G} is a pairwise Markov graph for P , then for non-overlapping sets of vertices A, B , and C and corresponding subvectors of \mathbf{X} respectively $\mathbf{X}_A, \mathbf{X}_B$, and \mathbf{X}_C

$$C \text{ separates } A \text{ and } B \Rightarrow \mathbf{X}_A \perp\!\!\!\perp \mathbf{X}_B \mid \mathbf{X}_C \quad (196)$$

If, for example, Figure 47 is a pairwise Markov graph for a distribution P for X_1, X_2, \dots, X_5 , we may conclude from implication (196) that

$$(X_1, X_2, X_5) \perp\!\!\!\perp (X_3, X_4) \quad \text{and} \quad X_2 \perp\!\!\!\perp X_5 \mid X_1$$

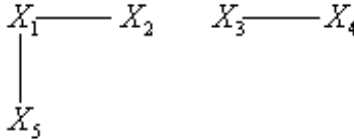


Figure 47: A pairwise Markov undirected graph for P .

Condition (196) says that graph separation implies conditional independence. An apparently stronger relationship would be the *equivalence* of the graphical and probabilistic conditions. For P a joint distribution for X_1, X_2, \dots, X_k and \mathcal{G} an undirected graph, we will say that P is **globally \mathcal{G} Markov** provided for all non-overlapping sets of vertices A, B , and C

$$C \text{ separates } A \text{ and } B \Leftrightarrow \mathbf{X}_A \perp\!\!\!\perp \mathbf{X}_B \mid \mathbf{X}_C$$

Then as it turns out,

$$P \text{ is globally } \mathcal{G} \text{ Markov} \Leftrightarrow \mathcal{G} \text{ is a pairwise Markov graph associated with } P$$

so that separation on a pairwise Markov graph is equivalent to conditional independence.

An important question is "What forms are possible for densities when P is globally \mathcal{G} Markov?" An answer is provided by the famous Hammersley-Clifford Theorem. This promises that if joint pmf $p_{\mathbf{X}}(\mathbf{x}) > 0$ for all \mathbf{x} and $\{C_1, C_2, \dots, C_m\}$ is the set of all maximal cliques for a pairwise Markov graph \mathcal{G} associated with P , then

$$p_{\mathbf{X}}(\mathbf{x}) \propto \prod_{i=1}^m \psi_i(\mathbf{x}_{C_i}) \quad (197)$$

for some functions $\psi_i(\cdot) > 0$. A potentially more natural but less parsimonious representation is that (if again joint pmf $p_{\mathbf{X}}(\mathbf{x}) > 0$ for all \mathbf{x})

$$p_{\mathbf{X}}(\mathbf{x}) \propto \prod_{i < j \text{ s.t. } X_i \sim X_j} \psi_{ij}(x_i, x_j) \quad (198)$$

for some functions $\psi_{ij}(\cdot) > 0$.

As for the directed case, the independence relationships implied by an associated Markov (undirected) graph enforce simplicity on a joint distribution P . Take, for example, the situation represented by Figure 46, supposing \mathbf{X} takes values in the small set $\{1, 2, 3\}^4$. In general, a pmf $p_{\mathbf{X}}(\mathbf{x})$ for this sample space is specified by $3^4 - 1 = 80$ probabilities. But the set of maximal cliques for the undirected graph in Figure 46 is $\{\{X_1, X_2, X_3\}, \{X_2, X_4\}\}$ so that form (197) promises that no more than $3^3 + 3^2 = 36$ values are needed to specify P that is globally or pairwise \mathcal{G} Markov for the undirected graph in Figure 46. Or, since there are 4 edges on the graph in Figure 46, form (198) promises that no more than $4 \cdot 3^2 = 36$ values are needed to specify P .

The same issues raised regarding the practical use of directed graphical models arise for undirected graphical models. What forms are appropriate for what kinds of problems? How does one infer the nature of the $\psi(\cdot)$ appearing in form (197) or (198) from training data, especially when some of the variables involved are latent/hidden/unobserved? Again, researchers who value this kind of modeling must obviously produce tractable and believable/well-motivated forms \mathcal{G} and corresponding forms for the $\psi(\cdot)$ that lead to effective specialized fitting methods for the kinds of training data they expect to encounter.

18.2.1 Restricted Boltzmann Machines

One particular version of undirected graphical modeling that has seen recent interest in machine learning applications is that of Restricted Boltzmann Machines. We will here consider the simplest of these models, where all variables are binary.⁴⁹ In this kind of model, nodes are arranged in two layers and

⁴⁹For some purposes 0/1 coding is most convenient and for others $-1/1$ coding of variables is most helpful. What follows can be read with either in mind. The class of models produced does not depend on this choice, only the interpretation of parameters of those models.

there are edges only between nodes on different layers, not between nodes in the same layer. One layer of nodes is called the "hidden layer" and the other is called the "visible layer." Typically the nodes in the visible layer correspond to (digital versions) of variables that are (at least at some cost) empirically observable, while the variables corresponding to hidden nodes are completely latent/unobservable and somehow represent some stochastic physical or mental mechanism. In addition, it is convenient in some contexts to think of visible nodes as being of two types, say belonging to a set V_1 or a set V_2 . For example, in a prediction context, the nodes in V_1 might encode " \mathbf{x} "/inputs and the nodes in V_2 might encode y /outputs. We'll use the naming conventions indicated in Figure 48.

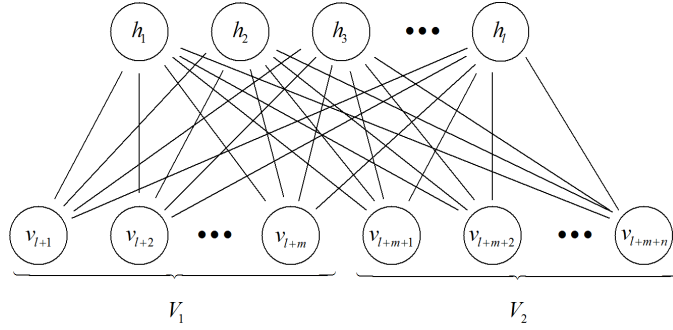


Figure 48: An undirected graph corresponding to a restricted Boltzmann machine with $l + m + n$ total nodes. l nodes are hidden and the $m + n$ visible nodes break into the two sets V_1 and V_2 .

Then for $l + m + n$ real parameters θ_k and $l(m + n)$ real parameters θ_{ij} (for $1 \leq i \leq l$ and $l + 1 \leq j \leq l + m + n$) the functions

$$\psi_{ij}(h_i, v_j | \boldsymbol{\theta}) = \exp \left(\left(\frac{\theta_i}{m+n} \right) h_i + \left(\frac{\theta_j}{l} \right) v_j + \theta_{ij} h_i v_j \right)$$

can be used in form (198) to produce a pmf for (\mathbf{h}, \mathbf{v}) for which Figure 48 provides a pairwise Markov graph. For this form

$$p(\mathbf{h}, \mathbf{v} | \boldsymbol{\theta}) \propto \exp \left(\sum_{i=1}^l \theta_i h_i + \sum_{j=l+1}^{l+m+n} \theta_j v_j + \sum_{i=1}^l \sum_{j=l+1}^{l+m+n} \theta_{ij} h_i v_j \right)$$

and thus

$$p(\mathbf{h}, \mathbf{v} | \boldsymbol{\theta}) = \frac{\exp \left(\sum_{i=1}^l \theta_i h_i + \sum_{j=l+1}^{l+m+n} \theta_j v_j + \sum_{i=1}^l \sum_{j=l+1}^{l+m+n} \theta_{ij} h_i v_j \right)}{\sum_{(\tilde{\mathbf{h}}, \tilde{\mathbf{v}})} \exp \left(\sum_{i=1}^l \theta_i \tilde{h}_i + \sum_{j=l+1}^{l+m+n} \theta_j \tilde{v}_j + \sum_{i=1}^l \sum_{j=l+1}^{l+m+n} \theta_{ij} \tilde{h}_i \tilde{v}_j \right)} \quad (199)$$

Let the normalizing constant that is the denominator on the right of display (199) be called $\gamma(\boldsymbol{\theta})$ and note the obvious fact that for these models

$$\ln(p(\mathbf{h}, \mathbf{v}|\boldsymbol{\theta})) = \sum_{i=1}^l \theta_i h_i + \sum_{j=l+1}^{l+m+n} \theta_j v_j + \sum_{i=1}^l \sum_{j=l+1}^{l+m+n} \theta_{ij} h_i v_j - \ln(\gamma(\boldsymbol{\theta})) \quad (200)$$

Observe that such a model can have as many as

$$l + m + n + l(n + m) = l + (l + 1)(n + m)$$

non-zero real parameters.

Because for typical applications 2^{l+n+m} can be very large (and thus computation of $\gamma(\boldsymbol{\theta})$ can be prohibitive) it is not common to simulate (\mathbf{h}, \mathbf{v}) directly. But since differences in the log probabilities in display (200) for the two possible values of an h_i or v_j are very simple, conditional probabilities for a single h_i or v_j are very easy to find, Gibbs sampling algorithms can be used to generate observations (\mathbf{h}, \mathbf{v}) from the pmf (199) for fixed $\boldsymbol{\theta}$. Similarly, it is easy to hold fixed part of (\mathbf{h}, \mathbf{v}) (for example corresponding to nodes in V_1) and simulate from conditional distributions via Gibbs sampling (for fixed $\boldsymbol{\theta}$), thereby enabling approximately optimal prediction of the rest of (\mathbf{h}, \mathbf{v}) (and, in particular, the nodes corresponding to V_2).

This all looks attractive/promising, but there are three fundamental difficulties related to these models, namely

1. the matter of fitting a vector of coefficients $\boldsymbol{\theta}$ is problematic,
2. the fact that many ("most") Boltzmann machines are nearly degenerate, concentrating their probability on relative few different vectors and
3. (related to 2.) many ("most") Boltzmann machines further have the unpleasant property that change of a value of a single entry of (\mathbf{h}, \mathbf{v}) can cause wide swings in the probability (199).

The first of these issues is well-recognized. The second and third seem far less well-appreciated and make these models often less than ideal for representing observed real variation in \mathbf{v} .

What will be available as training data for an RBM is some set of (potentially incomplete) vectors of values for visible nodes, say \mathbf{v}_i^* for $i = 1, \dots, N$ (that one will typically assume are independent and from some appropriate marginal distribution for visible vectors derived via summation from the overall joint distribution of values associated with all nodes visible and hidden). Notice now that even in a *hypothetical* case where one has "data" consisting of complete (\mathbf{h}, \mathbf{v}) pairs, the existence of the unpleasant normalizing constant $\gamma(\boldsymbol{\theta})$ would typically make optimization of a likelihood $\prod_{i=1}^N p(\mathbf{h}_i, \mathbf{v}_i|\boldsymbol{\theta})$ or log-likelihood $\sum_{i=1}^N \ln p(\mathbf{h}_i, \mathbf{v}_i|\boldsymbol{\theta})$ problematic. But the fact that one must sum out over (at least) all hidden nodes in order to get contributions to a likelihood or

loglikelihood makes the problem even more computationally difficult. That is, if an i th training case provides a complete visible vector \mathbf{v}_i , the corresponding likelihood term is the $\boldsymbol{\theta}$ marginal of that visible configuration

$$p(\mathbf{v}_i|\boldsymbol{\theta}) = \sum_{\tilde{\mathbf{h}}} p(\tilde{\mathbf{h}}, \mathbf{v}_i|\boldsymbol{\theta})$$

And the computational situation becomes even more unpleasant if an i th training case provides, for example, only values for variables corresponding to nodes in V_1 (say \mathbf{v}_{1i}), since the corresponding likelihood term is the marginal of only that visible configuration

$$p(\mathbf{v}_{1i}|\boldsymbol{\theta}) = \sum_{\tilde{\mathbf{h}} \text{ and } \tilde{\mathbf{v}}_2} p(\tilde{\mathbf{h}}, (\mathbf{v}_{1i}, \tilde{\mathbf{v}}_2)|\boldsymbol{\theta})$$

Substantial effort in computer science circles has gone into the search for "learning" algorithms aimed at finding parameter vectors $\boldsymbol{\theta}$ that produce large values of loglikelihoods based on N training cases (each term based on some marginal of $p(\mathbf{h}, \mathbf{v}|\boldsymbol{\theta})$ corresponding to a set of visible nodes). These seem to be mostly based on approximate stochastic gradient descent ideas and approximations to appropriate expectations based on short Gibbs sampling runs. Hinton's notion of "contrastive divergence" appears to be central to the most well known of these. Work of Kaplan *et al.* calls into question even the possibility of completely rational means of fitting Boltzmann machines by appeal to any standard statistical principles.

Beyond the fitting problem is the nature of many fitted Boltzmann Machines. These seem to typically seriously under-represent the kind of variability seen in \mathbf{v}_i s in training sets and have marginals $p(\mathbf{v}|\boldsymbol{\theta})$ that are highly sensitive to small (one-coordinate) changes in \mathbf{v} . For some theory in this direction, see again work of Kaplan *et al.* These issues seem to be real drawbacks to the use of RBMs in data modeling.

Some of what is termed "deep learning" seems to be based on the notion of generalizing RBMs to more than a single hidden layer and potentially employs visible layers on both the top and the bottom of an undirected graph. A cartoon of a deep learning network is given in Figure 49. The fundamental feature of the graph architecture here is that there are edges *only* between nodes in successive layers.

If the problem of how to fit parameters for a RBM is a practically difficult/??impossible? one, fitting a deep network model (based on some training cases consisting of values for variables associated with some of the visible nodes) is clearly going to be doubly problematic. What seems to be currently popular is some kind of "greedy"/"forward" sequential fitting of one set of parameters connecting two successive layers at a time, followed by generation of simulated values for variables corresponding to nodes in the "newest" layer and treating those as "data" for fitting a next set of parameters connecting two layers (and so on). But a deep learning network like that portrayed on Figure 49 compounds

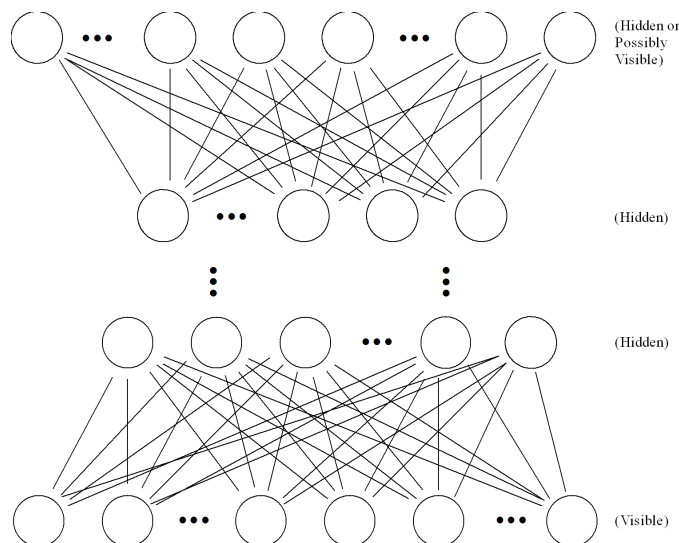


Figure 49: An hypothetical "deep" generalization of a RBM.

the kinds of fitting issues raised for RBMs and principled methods seem lacking. Further, degeneracy and instability issues like 2. and 3. above also are manifest.

If the fitting issues for a deep network were solvable for a given architecture and form of training data, some interesting possibilities for using a good "deep" model have been raised. For one, in a classification/prediction context, one might treat the bottom visible layer as associated with an input vector \mathbf{x} , and the top layer as also visible and associated with an output y or \mathbf{y} . In theory, training cases could consist of complete (\mathbf{x}, y) pairs, but they could also involve some incomplete cases, where y is missing, or part of \mathbf{x} is missing, or ... so on. (Once the training issue is handled, simulation of the top layer variables from inputs of interest used as bottom layer values again enables classification/prediction.)

As an interesting second possibility for using deep structures, one could consider architectures where the top and bottom layers are both visible and encode essentially (or even exactly⁵⁰) the same information and between them are several hidden layers with one having very few nodes (like, say, *two* nodes). If one can fit such a thing to training cases consisting of sets of values corresponding to visible nodes, one can simulate (again via Gibbs sampling) for a fixed set of "visible values" corresponding values at the few nodes serving as the narrow layer of the network. The vector of estimated marginal probabilities of a latent value 1 at those nodes might then in turn serve as a kind of pair of "generalized principal component" values for the set of input visible values. (Notice that in

⁵⁰I am not altogether sure what are the practical implications of essentially turning the kind of "tower" in Figure 49 into a "band" or flat bracelet that connects back on itself, the top hidden layer having edges directly to the bottom visible layer, but I see no obvious prohibition of this architecture.

theory these are possible to compute even for incomplete sets of visible values.)

19 Special Bayes Methods for Statistical Learning

19.1 Relevance Vector Machines

It is a theme that runs through much of modern prediction practice that shrinkage and smoothing and control of complexity of predictors is an essential part of finding effective versions of them. This is often accomplished by shrinking fitted parameter vectors of models toward regions of a parameter space corresponding to relatively simple sub-models.⁵¹ Some versions of partially Bayes prediction methodology meant to enforce such parameter induced simplicity/sparsity seem to go under the name "relevance vector machines." (Presumably this is about identifying a few "relevant" parameters of a model that should not be "zeroed out.") Here we consider some ideas in this direction for parameter vectors β that provide coefficients for linear forms upon which SEL or 0-1 loss predictors are to be built.

Suppose that for p predictors and N cases one creates from the p inputs a feature matrix \mathbf{H} using q real-valued functions $h_j(\mathbf{x})$. This could be the original data matrix \mathbf{X} in the case that $q = p$, a Gram matrix \mathbf{K} for a kernel \mathcal{K} (made with every $h_j(\mathbf{x}) = \mathcal{K}(\mathbf{x}, \mathbf{x}_j)$) in the case $q = N$, or simply a matrix of values for some set of basis functions h_1, \dots, h_q (each mapping $\mathbb{R}^p \rightarrow \mathbb{R}$). The idea is that a predictor of the output y is to be built on the product $(h_1(\mathbf{x}), \dots, h_q(\mathbf{x}))\beta$ for a q -vector of parameters β and that sparsity in the parameter vector (few entries of any appreciable size) corresponds to simplicity of the final predictor. The search for effective "relevance vector machines" is the search for prior distributions for β that encode sparsity.⁵²

Probably the simplest and most popular priors for this problem are so-called **spike and slab priors**. These are priors for β that for a large number B , a small positive constant α , and δ_0 the point mass distribution at 0 make its entries iid, with each β_j having the distribution

$$\alpha N(0, B^2) + (1 - \alpha)\delta_0 \quad \text{or} \quad \alpha U(-B, B) + (1 - \alpha)\delta_0$$

This kind of prior distribution is obviously symmetric in j and puts much of its prior probability on hyperplanes where some of the entries of β are 0. Cor-

⁵¹The support vector machine idea is an instance of this, where a relatively few "support vectors" of N possible training vectors are represented in formulas for optimal linear voting functions, because all others have coefficients that are "zeroed out" in the fitting process (this ultimately corresponding putting parameter vectors on 0-coordinate hyperplanes in an $(N + 1)$ -dimensional parameter space).

⁵²The fact that a posterior density is proportional to the product of the likelihood function and a prior density implies that the way to get posterior sparsity is to use prior distributions that put most of their mass on or near "simple sub-model" parts of the parameter space.

responding posterior distributions based on a training set typically concentrate on and near those hyperplanes.

The lasso development of Section 3.1.2 and the notion that penalization in normal data models is strongly related to use of priors whose log densities are proportional to the penalty functions suggests another possibility. That is the use of priors for β that for a single $\lambda > 0$ and an a single exponent $0 < r \leq 1$ make its entries iid with each β_j having density proportional to

$$\exp(-\lambda |\beta_j|^r)$$

(The $r = 1$ case is that of independent doubly exponential priors for the entries of β .) For large λ this symmetric prior again concentrates much of its mass near hyperplanes where some of the entries of β are 0.

A third sparsity-inducing prior for β employs a set of q additional hyper-parameters ϕ_1, \dots, ϕ_q and conditional on these makes the entries of β independent with $\beta_j \sim N(0, 1/\phi_j^2)$ (the *variance* is $1/\phi_j^2$). Then using independent proper hyper-priors

$$\phi_j \sim \Gamma(a, b) \quad \text{for small positive } a \text{ and } b$$

or independent Jeffreys improper hyper-priors with

$$-\frac{1}{2} \ln \phi_j \sim U(\mathcal{R})$$

or proper approximations to the Jeffreys priors like

$$-\frac{1}{2} \ln \phi_j \sim U(-B, B) \quad \text{for large } B$$

often posteriors for many of the ϕ_j have large mass far from 0 and correspondingly encode large concentration of mass for many of the β_j near 0.

The third possibility above has some corresponding analytical results that can be used to approximate posteriors, but the most direct way of processing a training set and prior distribution to produce usable posterior results is through the use of standard Bayes MCMC software. In SEL prediction problems, one can for example combine a prior distribution for β with a likelihood derived from a model for independent

$$y_i \sim N(\beta_0 + (h_1(\mathbf{x}_i), \dots, h_q(\mathbf{x}_i))\beta, \sigma^2)$$

(and appropriate priors for β_0 and σ^2). In 2-class classification problems (with 0-1 coding) one can combine a prior distribution for β with a likelihood derived from a model for independent

$$y_i \sim \text{Bernoulli} \left(\frac{1}{1 + \exp(-(\beta_0 + (h_1(\mathbf{x}_i), \dots, h_q(\mathbf{x}_i))\beta))} \right)$$

(and an appropriate prior for β_0). In both cases, standard Bayes software can be used to identify a (typically sparse) high-posterior density parameter vector $\hat{\beta}$ and then a sensible SEL predictor

$$\hat{f}(\mathbf{x}) = \hat{\beta}_0 + (h_1(\mathbf{x}), \dots, h_q(\mathbf{x}))\hat{\beta}$$

in the first case and a sensible 0-1 loss classifier

$$\hat{f}(\mathbf{x}) = I \left[\hat{\beta}_0 + (h_1(\mathbf{x}), \dots, h_q(\mathbf{x})) \hat{\beta} > 0 \right]$$

in the second.

19.2 Dirichlet and Data-Derived Priors for Prediction Based on Normal Mixture Models

A basic reality of high-dimensional (large p) prediction is that it is rare to encounter a problem where a single simple relationship between input \mathbf{x} and output y holds across a large input space. What one *might* hope for, however, is to find regions in \mathfrak{R}^p where different simple relationships hold and to more or less tie those relationships together (across the relevant part of \mathfrak{R}^p) probabilistically. Work of Lanker, Ryan, Culp, Vardeman, and Morris has been built on this idea and (multivariate) normal mixture models. This section outlines some of that.

For

$$\begin{pmatrix} y \\ \mathbf{x} \end{pmatrix} \sim \text{MVN}_{p+1}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

$E[y|\mathbf{x}]$ is a very simple linear function of \mathbf{x} . So if a K -vector of (mixture) probabilities $\boldsymbol{\pi} = (\pi_1, \pi_2, \dots, \pi_K)$ (of course with $\sum_{k=1}^K \pi_k = 1$) and K means $\boldsymbol{\mu}_k$ and covariance matrices $\boldsymbol{\Sigma}_k$ together specify a mixture distribution and

$$\begin{pmatrix} y \\ \mathbf{x} \end{pmatrix} \sim \sum_{k=1}^K \pi_k \text{MVN}_{p+1}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \quad (201)$$

then for $p(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ the k th (marginal) component density of \mathbf{x} and $E_k[y|\mathbf{x}]$ the k th (linear) conditional mean function, then

$$E[y|\mathbf{x}] = \sum_{k=1}^K \left(\frac{\pi_k p(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{k=1}^K \pi_k p(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)} \right) E_k[y|\mathbf{x}] \quad (202)$$

where the multiplier of conditional mean k (call it $\pi_k(\mathbf{x})$) is the conditional probability that \mathbf{x} is from the corresponding component of the mixture. Armed with the mixture probabilities, means, and covariance matrices, formula (202) provides a "locally" (where some $\pi_k(\mathbf{x})$ is essentially 1) simple (linear) SEL predictor for y . Of course, one doesn't know parameter values needed to compute the predictor except through a training set \mathbf{T} .

It turns out that it often does not work well in practice to simply estimate the parameters of the mixture distribution (201) from a training set and plug those estimates into form (202) to make a SEL predictor for y . But what Lanker *et al.* have found to be effective is based on a Bayes model and use of latent "component identity" variables. That is, for a training set⁵³ $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)$ invent N corresponding latent variables k_1, \dots, k_N each

⁵³Without loss of generality, suppose that every x_j and y has been standardized.

taking values in $\{1, 2, \dots, K\}$ iid with marginal distribution specified by $\boldsymbol{\pi}$. If conditioned on k_i each

$$\begin{pmatrix} y_i \\ \mathbf{x}_i \end{pmatrix} \sim \text{MVN}_{p+1}(\boldsymbol{\mu}_{k_i}, \boldsymbol{\Sigma}_{k_i})$$

the training set has N observations that are iid according to the mixture distribution (201). Then for a prior distribution

$$\boldsymbol{\pi} \sim \text{Dirichlet}(\alpha_1, \dots, \alpha_K) \quad (203)$$

(all α_k s the same works well in practice) and independent priors

$$\boldsymbol{\mu}_k \stackrel{\text{iid}}{\sim} g_1(\cdot) \quad \text{and} \quad \boldsymbol{\Sigma}_k \stackrel{\text{iid}}{\sim} g_2(\cdot) \quad (204)$$

MCMC algorithms for sampling from the posterior distribution of the parameters of the mixture and the latent variables are easy to find. Iterates of the parameters vectors from such an algorithm produce iterates of the functional form (202) and averaging across iterates can produce a workable predictor. This works by essentially picking out "the right" regions and linear functions where linearity of prediction is warranted and by identifying "the right" number of components (less than or equal to K) to assign appreciable entries in $\boldsymbol{\pi}$. But it seems that there are three requirements for the forms $g_1(\cdot)$ and $g_2(\cdot)$ in order for this program to be effective. These are that 1) some kind of conjugacy is needed in order to make Gibbs sampling applicable and the method practical, 2) "locations" for $g_1(\cdot)$ need to be "right" and "scales" for $g_2(\cdot)$ need to be flexible, and 3) neither "flat"/uninformative nor very "sharp"/informative distributions work well for forms $g_1(\cdot)$ and $g_2(\cdot)$.

For purposes of making an effective predictor (not for purposes of a philosophically "proper" Bayes analysis) it proves effective to employ $g_1(\cdot)$ derived from the training set. One can effectively use a multivariate density estimate based on the observed vectors in the training set and a spherical normal kernel

$$g_1(\boldsymbol{\mu}) = \frac{1}{N} \sum_{i=1}^N p\left(\boldsymbol{\mu} \mid \begin{pmatrix} y_i \\ \mathbf{x}_i \end{pmatrix}, \sigma^2 \mathbf{I}\right)$$

for an appropriate bandwidth σ . (One can simulate from this prior by picking a training case at random and adding to it a $\text{MVN}_{p+1}(\mathbf{0}, \sigma^2 \mathbf{I})$ random perturbation.)

Further (still for purposes of making an effective predictor) it proves effective to employ for $g_2(\cdot)$ an equally weighted mixture of inverse Wishart densities with corresponding "means" $\gamma^2 \mathbf{I}$ and minimum (namely $p+3$) degrees of freedom for $\gamma \in \{.01, .02, \dots, 1.00\}$. This mixture prior allows different scales for different components of form (201) and gives a group of training cases i with common k_i maximum effect on what values of $\boldsymbol{\Sigma}_{k_i}$ have large posterior probability (tending to make $\boldsymbol{\Sigma}_{k_i}$ look like the group sample covariance matrix).

The product of $g_1(\cdot)$ and $g_2(\cdot)$ is then a mixture of joint densities conjugate in the one-sample multivariate normal problem and is thus easy to handle in

Gibbs sampling. Gibbs updating of $\boldsymbol{\pi}$ is similarly easy using the k_i s, and Gibbs updates of those are easily handled because they are discrete. In all, the data-dependent "prior" distribution specified in displays (203) and (204) is computationally attractive and leads to effective SEL prediction.

19.3 Bayes Mixture Analyses for Binary Vectors

The foregoing material on Dirichlet priors for multivariate normal mixtures has an interesting parallel in a class of symmetric component models for (binary) vectors $\boldsymbol{x} \in \{0, 1\}^p$. We here briefly discuss recent work of Chakraborty and Vardeman in this direction.

This begins from the exponential family of distributions on $\{0, 1\}^p$ defined for parameters $\boldsymbol{\mu} \in \{0, 1\}^p$ and $\gamma \in (0, 1)$ by the pmf

$$p(\boldsymbol{x}|\boldsymbol{\mu}, \gamma) = \left(\frac{1 - \gamma}{1 - \gamma^p} \right) \frac{\gamma^{\|\boldsymbol{x} - \boldsymbol{\mu}\|^2}}{\binom{p}{\|\boldsymbol{x} - \boldsymbol{\mu}\|^2}} \quad (205)$$

The parameter $\boldsymbol{\mu}$ is a "central value" for the distribution and $\|\boldsymbol{x} - \boldsymbol{\mu}\|^2$ is simply the number of coordinates at which \boldsymbol{x} and $\boldsymbol{\mu}$ differ. The total probability assigned (uniformly) to those \boldsymbol{x} which differ from $\boldsymbol{\mu}$ in m coordinates decreases geometrically in m , and γ thus functions as a "spread" parameter for the distribution. (Small values of γ yield distributions highly concentrated at $\boldsymbol{\mu}$ and large γ s have corresponding distributions approximately uniform on $\{0, 1\}^p$.)

Since an arbitrary distribution for binary vectors $\boldsymbol{x} \in \{0, 1\}^p$ is defined by 2^p probabilities, it is obviously possible to approximate any distribution for binary vectors with a mixture of distributions with pmfs (205) to any desired degree of precision. What is really more interesting is the possibility of describing distributions for binary vectors in many applications as mixtures with *relatively few components*, the $\boldsymbol{\mu}$ s representing modes or representative cases in the space $\{0, 1\}^p$ and the corresponding γ s controlling how many changes of coordinates away from these modes are likely.

For local notational convenience, call a distribution on $\{0, 1\}^p$ of the form (205) the $\text{SBEF}_p(\boldsymbol{\mu}, \gamma)$ distribution (for Symmetric Binary Exponential Family). So a K -vector of (mixture) probabilities $\boldsymbol{\pi} = (\pi_1, \pi_2, \dots, \pi_K)$ (of course with $\sum_{k=1}^K \pi_k = 1$) and K centers $\boldsymbol{\mu}_k$ and spread parameters γ_k together specify a mixture distribution

$$\sum_{k=1}^K \pi_k \text{SBEF}_p(\boldsymbol{\mu}_k, \gamma_k) \quad (206)$$

that is a potentially useful basis of modeling and inference.

As in the MVN case of the previous discussion, it proves helpful to invent and use latent "component identity" variables. That is, for a training set $\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N$, invent N corresponding variables k_1, k_2, \dots, k_N each taking values in $\{1, 2, \dots, K\}$ iid with marginal distribution specified by $\boldsymbol{\pi}$. If conditioned

on k_i each

$$\mathbf{x}_i \sim \text{SBEF}_p(\boldsymbol{\mu}_{k_i}, \gamma_{k_i})$$

the training set has N observations that are iid according to the mixture distribution (206). Then for a prior distribution

$$\boldsymbol{\pi} \sim \text{Dirichlet}(\alpha_1, \dots, \alpha_K)$$

(all α_k s the same works well in practice) and independent priors

$$\boldsymbol{\mu}_k \stackrel{\text{iid}}{\sim} \text{U}(\{0, 1\}^p) \quad \text{and} \quad \gamma_k \stackrel{\text{iid}}{\sim} g(\cdot) \quad (207)$$

where

$$g(\gamma) \propto \sqrt{\frac{1 - (p+1)^2 \gamma^p + 2p(p+2) \gamma^{p+1} - (p+1)^2 \gamma^{p+2} + \gamma^{2p+2}}{\gamma(1-\gamma^{p+1})^2(1-\gamma)^2}}$$

(this latter is a univariate Jeffreys prior for γ in a model where $\boldsymbol{\mu}$ is known to be $\mathbf{0}$) MCMC algorithms for sampling from the posterior distribution of the parameters of the mixture and the latent variables are easy to find. Iterates of the parameters vectors from such an algorithm produce a number of useful quantities. $\boldsymbol{\mu}$ s that appear with highest frequency in the iterates are candidates for modes/representative cases of binary vectors. The average across iterates of the probability assigned by the mixture to a value \mathbf{x} serves as a posterior mean for the probability of that value. And for any two indices i and i' , the relative frequency among iterates with which $k_i = k_{i'}$ serves as an approximate posterior probability that case i and case i' share a common origin and can be used as bases for clustering cases, much as was suggested for Zhou's Bayesian SOM and Chakraborty's Bayes biclustering.

Part VII

Appendices

A Exercises

A.1 Section 1.2 Exercises

These are exercises intended to provide intuition that data in \mathfrak{R}^p are necessarily "sparse." The realities are that \mathfrak{R}^p is "huge" and for p at all large, "filling up" even a small part of it with data points is effectively impossible and our intuition about distributions in \mathfrak{R}^p is very poor.

1. (6HW-11) Let $Q_p(t)$ and $q_p(t)$ be respectively the χ_p^2 cdf and pdf. Consider the MVN $_p(\mathbf{0}, \mathbf{I})$ distribution and $\mathbf{Z}_1, \mathbf{Z}_2, \dots, \mathbf{Z}_N$ iid with this distribution. With

$$M = \min \{ \|\mathbf{Z}_i\| \mid i = 1, 2, \dots, N \}$$

write out a one-dimensional integral involving $Q_p(t)$ and $q_p(t)$ giving EM. Evaluate this mean for $N = 100$ and $p = 1, 5, 10$, and 20 either numerically or using simulation.

2. (6HW-13) For each of $p = 1, 5, 10$, and 20, generate at least 1000 realizations of pairs of points \mathbf{x} and \mathbf{z} as iid uniform over the p -dimensional unit ball (the set of \mathbf{x} with $\|\mathbf{x}\| \leq 1$). Compute (for each p) the sample average distance between \mathbf{x} and \mathbf{z} . (For $\mathbf{Z} \sim \text{MVN}_p(\mathbf{0}, \mathbf{I})$ independent of $U \sim U(0, 1)$, $\mathbf{x} = (U^{1/p} / \|\mathbf{Z}\|) \mathbf{Z}$ is uniformly distributed in the unit ball in \mathbb{R}^p .)

3. (5HW-14) For each of $p = 10, 20, 50, 100, 500$, and 1000, make $n = 10,000$ draws of distances between pairs of independent points uniform in the cube $[0, 1]^p$. Use these to make 95% confidence limits for the ratio

$$\frac{\text{mean distance between two random points in the cube}}{\text{maximum distance between two points in the cube}}$$

4. (5HW-14) For each of $p = 10, 20, 50$, make $n = 10,000$ random draws of $N = 100$ independent points uniform in the cube $[0, 1]^p$. Find for each sample of 100 points, the distance from the first point drawn to the 5th closest point of the other 99. Use these to make 95% confidence limits for the ratio

$$\frac{\text{mean diameter of a 5-nearest neighbor neighborhood if } N = 100}{\text{maximum distance between two points in the cube}}$$

5. (5HW-14) What fraction of random draws uniform from the unit cube $[0, 1]^p$ lie in the "middle part" of the cube $[\epsilon, 1 - \epsilon]^p$, for a small positive number ϵ ?

The next 3 problems are based on nice ideas taken from Giraud's book.

6. (6HW-15) For $p = 2, 10, 100$, and 1000, draw samples of size $N = 100$ from the uniform distribution on $[0, 1]^p$. Then for every $(\mathbf{x}_i, \mathbf{x}_j)$ pair with $i < j$ in one of these samples, compute the Euclidean distance between the two points, $\|\mathbf{x}_i - \mathbf{x}_j\|$. Make a histogram (one p at a time) of these $\binom{100}{2}$ distances. What do these suggest about how well "local" prediction methods (that rely only on data points (\mathbf{x}_i, y_i) with \mathbf{x}_i "near" \mathbf{x} to make predictions about y at \mathbf{x}) can be expected to work?

7. (6HW-15) Consider finding a lower bound on the number of points \mathbf{x}_i (for $i = 1, 2, \dots, N$) required to "fill up" $[0, 1]^p$ in the sense that no point of $[0, 1]^p$ is Euclidean distance more than ϵ away from some \mathbf{x}_i .

The p -dimensional volume of a ball of radius r in \Re^p is

$$V_p(r) = \frac{\pi^{p/2}}{\Gamma(p/2 + 1)} r^p$$

and Giraud notes that it can be shown that as $p \rightarrow \infty$

$$\frac{V_p(r)}{\left(\frac{2\pi\epsilon r^2}{p}\right)^{p/2} (p\pi)^{-1/2}} \rightarrow 1$$

Then, if N points can be found with corresponding ϵ -balls covering the unit cube in \Re^p , the total volume of those balls must be at least 1. That is

$$NV_p(\epsilon) \geq 1$$

What then are approximate lower bounds on the number of points required to fill up $[0, 1]^p$ to within ϵ for $p = 20, 50$, and 200 , and $\epsilon = 1, .1$, and $.01$? (Giraud notes that the $p = 200$ and $\epsilon = 1$ lower bound is larger than the estimated number of particles in the universe.)

8. (6HW-15) Giraud points out that for large p , most of $MVN_p(\mathbf{0}, \mathbf{I})$ probability is "in the tails." For $q_p(\mathbf{x})$ the $MVN_p(\mathbf{0}, \mathbf{I})$ pdf and $0 < \delta < 1$ let

$$B_p(\delta) = \{\mathbf{x} | q_p(\mathbf{x}) \geq \delta q_p(\mathbf{0})\} = \{\mathbf{x} | \|\mathbf{x}\|^2 \leq 2 \ln(\delta^{-1})\}$$

be the "central"/"large density" part of the multivariate standard normal distribution.

a) Using the Markov inequality, show that the probability assigned by the multivariate standard normal distribution to the region $B_p(\delta)$ is no more than $1/\delta 2^{p/2}$.

b) What then is a lower bound on the radius of a ball at the origin (call it $r(p)$) required so that the multivariate standard normal distribution places probability $.5$ in that ball? What is an upper bound on the ratio $q_p(\mathbf{x})/q_p(\mathbf{0})$ outside the ball with radius that lower bound? Plot these bounds as functions of p for $p \in [1, 500]$.

A.2 Section 1.3 Exercises

1. (6HW-17)

a) Argue carefully that for inherently non-negative response y with loss

$$L(\hat{y}, y) = \left[\ln \left(\frac{\hat{y} + 1}{y + 1} \right) \right]^2$$

a theoretically optimal predictor is

$$f(\mathbf{x}) = \exp(E[\ln(y + 1) | \mathbf{x}]) - 1$$

b) The Zillow Kaggle game for predicting (positive) house prices used the loss function

$$L(\hat{y}, y) = (\ln \hat{y} - \ln y)^2 = \left(\ln \frac{\hat{y}}{y} \right)^2$$

Identify the function of \mathbf{x} , call it $f(\mathbf{x})$, that based on a joint distribution P for (\mathbf{x}, y) optimizes

$$EL(g(\mathbf{x}), y)$$

over choices of function $g(\mathbf{x})$.

2. (6HW-13) Consider the loss function $L(\hat{y}, y) = (1 - y\hat{y})_+$ for y taking values in $\{-1, 1\}$ and prediction \hat{y} in \mathfrak{R} . Suppose that $P[y = 1] = p$. Write out the expected (over the randomness in y) loss of prediction \hat{y} . Plot this as a function of \hat{y} for the cases where first $p < .5$ and then $p > .5$. (These are continuous functions that are linear on the intervals $(-\infty, -1)$, $(-1, 1)$, and $(1, \infty)$.) What is an optimal choice of \hat{y} (depending upon p)?

3. (5E1-18) Consider predictors of $y \in \mathfrak{R}$ for $x \in [0, 1]$ based on linear combinations of the small set of "features" (functions of x)

$$h_1(x) = I\left[0 \leq x \leq \frac{1}{3}\right], h_2(x) = I\left[\frac{1}{3} < x \leq \frac{2}{3}\right], \text{ and } h_3(x) = I\left[\frac{2}{3} < x \leq 1\right]$$

and the very small training set

Case (i)	1	2	3	4	5	6
y_i	0	4	10	12	6	10
x_i	.1	.3	.4	.6	.7	.9

a) Without bothering to center y , consider using OLS to fit a predictor for y of the form $\hat{f}(x) = b_1 h_1(x) + b_2 h_2(x) + b_3 h_3(x)$ to the training set. Evaluate the LOOCV RMSPE for this kind of predictor.

b) Center the response (leaving the input as is) and fit a predictor (for centered response) of the form $\hat{f}(x) = b_1 h_1(x) + b_2 h_2(x) + b_3 h_3(x)$ via penalized least squares with penalty $\lambda(b_1^2 + b_2^2 + b_3^2)$ for $\lambda > 0$. (Give formulas for the 3 coefficients.)

4. (5E1-18) Use the same training set as in Problem 3 above and without bothering to center y , find the 1-nn SEL predictor for y , say $\hat{f}^{1\text{-nn}}(x)$, and evaluate its LOOCV MSPE. (Specify values of the predictor for all $x \in [0, 1]$ except where there are "ties.")

5. (5HW-18) Consider the Ames House Price dataset and possible predictors of Price. In particular, consider the $p = 4$ inputs Size, Fireplace, Basementbath, and Land. There are, of course, $2^4 = 16$ possible multiple linear regression predictors to be built from these features (including the one with no covariates employed). Use both LOOCV and repeated 8-fold cross-validation

implemented through `caret train()` to compare these 16 predictors in terms of cross-validation root mean squared prediction errors.

6. (5HW-18) Consider the famous "Glass Identification" dataset of German on the UCI Machine Learning Data Repository and k -nn classification between glass Types 1 and 2.

a) Use both LOOCV and repeated 10-fold cross-validation to find what you believe to be a best number of neighbors for this prediction task.

b) For your choice of k =number of neighbors in **a**), the variable $t(\mathbf{x})$ =number of Type 2 cases in the k -nearest neighborhood of \mathbf{x} can take values $0, 1, 2, \dots, k$. The nearest neighbor classifier classifies to Type 2 if $t(\mathbf{x}) \geq k/2$. This is based on $N = 146$ training cases of which 70 are of glass Type 1 and 76 are of glass Type 2. Suppose that you want to use $\pi_1 = .7$ and $\pi_2 = .3$ and 0-1 loss. How, if at all, would you modify the ordinary k -nn classifier?

7. (5HW-14) Consider 4 different continuous distributions on the 2-dimensional unit square $(0, 1)^2$ with densities on that space

$$p((x_1, x_2) | 1) = 1, p((x_1, x_2) | 2) = x_1 + x_2, p((x_1, x_2) | 3) = 2x_1,$$

$$\text{and } p((x_1, x_2) | 4) = x_1 - x_2 + 1$$

For a 0-1 loss $K = 4$ classification problem, find explicitly and make a plot showing the 4 regions in the unit square where an optimal classifier f has $f(\mathbf{x}) = k$ (for $k = 1, 2, 3, 4$) first if $\boldsymbol{\pi} = (\pi_1, \pi_2, \pi_3, \pi_4)$ is $(.25, .25, .25, .25)$ and then if it is $(.2, .2, .3, .3)$.

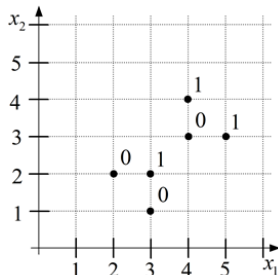
8. (5E1-14) Suppose that (unbeknownst to a statistical learner) $x \sim U(0, 1)$ and $E[y|x] = I[.45 < x < .55]$ (that is, the conditional mean of y given x is 1 when $.45 < x < .55$ and is 0 otherwise). A 3-nearest-neighbor predictor, \hat{f}_N , is based on N data pairs, and $\hat{f}_N(.5)$ has conditional means given the values of the inputs in the training set:

0	if no x_i is in $(.45, .55)$
1/3	if one x_i is in $(.45, .55)$
2/3	if two x_i s are in $(.45, .55)$
1	if three or more x_i s are in $(.45, .55)$

What is the value of the bias of the nearest neighbor predictor at .5? Does this bias go to 0 as N gets big? Argue carefully one way or the other.

9. (6E1-19) Below is a representation of an $N = 6$ toy 2-class classification training set with $p = 2$. Find the 0-1 loss LOOCV error rate for the 3-nearest-

neighbor classifier based on these training cases.



10. (6HW-11) Consider SEL prediction. Suppose that in a very simple problem with $p = 1$, the distribution P for the random pair (x, y) is specified by

$$x \sim U(0, 1) \text{ and } y|x \sim N(x^2, (1+x))$$

($(1+x)$ is the conditional variance of the output). Further, consider two possible sets of functions $S = \{g\}$ for use in creating predictors of y , namely

1. $S_1 = \{g|g(x) = a + bx \text{ for real numbers } a, b\}$, and
2. $S_2 = \left\{ g|g(x) = \sum_{j=1}^{10} a_j I \left[\frac{j-1}{10} < x \leq \frac{j}{10} \right] \text{ for real numbers } a_j \right\}$

Training data are N pairs (x_i, y_i) iid P . Suppose that the fitting of elements of these sets is done by

1. OLS (simple linear regression) in the case of S_1 , and
2. according to

$$\hat{a}_j = \begin{cases} \bar{y} & \text{if no } x_i \in \left(\frac{j-1}{10}, \frac{j}{10} \right] \\ \frac{1}{\#x_i \in \left(\frac{j-1}{10}, \frac{j}{10} \right]} \sum_{\substack{i \text{ with} \\ x_i \in \left(\frac{j-1}{10}, \frac{j}{10} \right]}} y_i & \text{otherwise} \end{cases}$$

in the case of S_2 ,

to produce predictors \hat{f}_1 and \hat{f}_2 .

a) Find (analytically) the functions g^* for the two cases. Use them to find the two expected squared model biases $E^x (E[y|x] - g^*(x))^2$. How do these two compare?

b) For the second case, find an analytical form for $E^T \hat{f}_2$ and then for the average squared fitting bias $E^x \left(E^T \hat{f}_2(x) - g_2^*(x) \right)^2$. (Hints: What is the conditional distribution of the y_i given that no $x_i \in \left(\frac{j-1}{10}, \frac{j}{10} \right]$? What is the conditional mean of y given that $x \in \left(\frac{j-1}{10}, \frac{j}{10} \right]$?)

c) For the first case, simulate at least 1000 training datasets of size $N = 100$ and do OLS on each one to get corresponding \hat{f}_1 s. Average those to get an approximation for $E^{\mathbf{T}} \hat{f}_1$. (If you can do this analytically, so much the better!) Use this approximation and analytical calculation to find the average squared fitting bias $E^x \left(E^{\mathbf{T}} \hat{f}_1(x) - g_1^*(x) \right)^2$ for this case.

d) How do your answers for **b)** and **c)** compare for a training set of size $N = 100$?

e) Use whatever combination of analytical calculation, numerical analysis, and simulation you need to use (at every turn preferring analytics to numerics to simulation) to find the expected prediction variances $E^x \text{Var}^{\mathbf{T}} \left(\hat{f}(x) \right)$ for the two cases for training set size $N = 100$.

f) In sum, which of the two predictors here has the best value of Err for $N = 100$?

11. (6HW-11) Two files with respectively $N = 100$ and then $N = 1000$ pairs (x_i, y_i) generated according to P in Problem 10 above are provided with these notes. Use 10-fold cross validation to see which of the two predictors in Problem 10 looks most likely to be effective. (The datasets will not be sorted, so you may treat successively numbered groups of 1/10th of the training cases as your $K = 10$ randomly created pieces of the training set.)

12. (5HW-14) Again consider SEL prediction. Suppose that (unknown to a statistician) a mechanism generates iid data pairs (x, y) according to the following model:

$$\begin{aligned} x &\sim U(-\pi, \pi) \\ y|x &\sim N\left(\sin(x), .25(|x| + 1)^2\right) \end{aligned}$$

(The conditional *variance* is $.25(|x| + 1)^2$.)

a) What is an absolutely minimum value of Err possible regardless what training set size, N , is available and what fitting method is employed?

b) What linear function of x (which $g(x) = a + bx$) has the smallest "average squared bias" as a predictor for y ? What cubic function of x (which $g(x) = a + bx + cx^2 + dx^3$) has the smallest average squared bias as a predictor for y ? Is the set of cubic functions big enough to eliminate model bias in this problem?

13. (5HW-14) An $N = 100$ dataset generated according to the model of Problem 12 is provided with these notes. Use 10-fold cross validation (use the 1st ten points as the fold, the 2nd 10 points as the second, etc.) based on the dataset to choose among the following methods of prediction for this scenario:

- polynomial regressions of orders 0, 1, 2, 3, 4, and 5,
- regressions using sets of predictors $\{1, \sin x, \cos x\}$ and $\{1, \sin x, \cos x, \sin 2x, \cos 2x\}$, and

- a regression with the set of predictors

$$\{1, x, x^2, x^3, x^4, x^5, \sin x, \cos x, \sin 2x, \cos 2x\}$$

(Use ordinary least squares fitting.) Which predictor looks best on an empirical basis? Knowing how the data were generated (an unrealistic luxury) which methods here are without model bias?

- 14. (5E1-14)** Consider a joint pdf (for $(x, y) \in (0, 1) \times (0, \infty)$) of the form

$$p(x, y) = \frac{1}{x^2} \exp\left(-\frac{y}{x^2}\right) \text{ for } 0 < x < 1 \text{ and } 0 < y$$

($x \sim U(0, 1)$ and conditional on x , the variable y is exponential with mean x^2 .)

a) Find the linear function of x (say $\alpha + \beta x$) that minimizes $E(y - (\alpha + \beta x))^2$. (The averaging is over the joint distribution of (x, y) . Find the optimizing intercept and slope.)

b) Suppose that a training set consists of N data pairs (x_i, y_i) that are independent draws from the distribution specified above, and that least squares is used to fit a predictor $\hat{f}_N(x) = a_N + b_N x$ to the training data. Suppose that it's possible to argue that the least squares coefficients a_N and b_N converge (in a proper probabilistic sense) to your optimizers from **a**) as $N \rightarrow \infty$. Then for large N , about what value of (SEL) training error do you expect to observe under this scenario?

- 15. (5E1-16)** Unknown to statistical learners in a $p = 1$ SEL prediction problem, $x \sim U(0, 6)$ and $y|x \sim N(x - 3, (x + 1)^2)$ (the conditional *variance* is $(x + 1)^2$). A statistical learner uses a class of predictors S consisting of all functions of the form $g_{a,b}(x) = a \cdot I[x < 2] + b \cdot I[x \geq 2]$.

a) In this context, what are

- the minimum expected loss possible,
- the best element of S , and
- the learner's modeling penalty?

b) Suppose that based on a training set of size $N = N_1 + N_2$ where N_1 is the count of x_i that are less than 2 and N_2 is the count of x_i that are at least 2, the fitting procedure used is to take⁵⁴ $\hat{a} = \bar{y}_1$ and $\hat{b} = \bar{y}_2$ (with the understanding that if $N_1 = 0$ then $\hat{a} = 0$ and if $N_2 = 0$ then $\hat{b} = 0$). Write an explicit expression for the fitting penalty here. (Hint: What is the distribution of N_1 ? Given that an x_i is less than 2, what are the mean and variance of y ? Given that an x_i is at least 2, what are the mean and variance of y ?)

c) Suppose that a second statistical learner uses predictors $h_{c,d}(x) = c \cdot I[x < 3] + d \cdot I[x \geq 3]$. A best such predictor is in fact $h_{-1.5, 1.5}(x) = -\frac{3}{2}I[x < 3] +$

⁵⁴In the obvious way, \bar{y}_1 is the sample mean output for inputs $x_i < 2$ and \bar{y}_2 is the sample mean output for inputs $x_i \geq 2$.

$\frac{3}{2}I[x \geq 3]$. Find a linear combination of the best element of S you identified in **a**) and this best predictor available to the second learner that is better than either individual predictor.

16. (6HW-13) Using the datasets provided with these notes carry out the steps of Problems 10 and 11 above supposing that the distribution P for the random pair (x, y) is specified by

$$x \sim U(0, 1) \text{ and } y|x \sim \text{Exp}(x^2)$$

(the exponential *mean* is x^2).

17. (6HW-15) Using the datasets provided with these notes carry out the steps of Problems 10 and 11 above supposing that the distribution P for the random pair (x, y) is specified by

$$x \sim U(0, 1) \text{ and } y|x \sim N\left((3x - 1.5)^2, (3x - 1.5)^2 + .2\right)$$

(the Gaussian *variance* is $(3x - 1.5)^2 + .2$).

18. (6E1-15) Consider a SEL prediction problem where $p = 1$ and the class of functions used for prediction is (the set of constant functions) $S = \{h|h(x) = c \forall x \text{ and some } c \in \mathfrak{R}\}$. Suppose that in fact

$$x \sim U(0, 1), \text{ E}[y|x] = ax + b, \text{ and } \text{Var}[y|x] = dx^2 \text{ for some } d > 0$$

a) Under this model, what is the best element of S , say g^* , for predicting y ? Use this to find the average squared model bias in this problem.

b) Suppose that based on an iid sample of N points (x_i, y_i) , fitting is done by least squares (and thus the predictor $\hat{f}(x) = \bar{y}$ is employed). What is the average squared fitting bias in this case?

c) What is the average prediction error, Err , when the predictor in **b)** is employed?

19. (6HW-17) Consider a toy 2-class classification model for $p = 1$, where $x|y = 0$ is $N(0, 1)$, $x|y = 1$ is $N(1, (.5)^2)$ (the standard deviation is $.5$), and $P[y = 0] = .5 = P[y = 1]$.

a) Compute and plot the function $P[y = 1|x]$.

b) Identify the optimal 0-1 loss classifier and the best possible expected loss/error rate in this classification problem. (This is a numerical problem.)

c) Consider the set of "linear" classifiers

$$S = \{I[x < c] | c \in \mathfrak{R}\} \cup \{I[x > c] | c \in \mathfrak{R}\}$$

(that make one cut in the real numbers at c and classify one way to the left of c and the other way to the right of c). Plot as functions of c the risks

$$\text{E}(I[y = 0]I[x < c] + I[y = 1]I[x > c])$$

for classifiers of the form $I[x < c]$ and

$$E(I[y = 0]I[x > c] + I[y = 1]I[x < c])$$

for classifiers of the form $I[x > c]$. What is the best element of S (say, g^*) and then what is the "modeling penalty" associated with using the class of predictors/classifiers S (the difference between the optimal error rate and the error rate for g^*)?

d) Suppose that for a training set of size $N = 100$ (generated at random from the distribution described in the preamble of this problem), one will choose a cut point \hat{c} half way between two consecutive sorted x_i values minimizing

$$\min[\#\{y_i = 0|x_i < c\} + \#\{y_i = 1|x_i > c\}, \#\{y_i = 1|x_i < c\} + \#\{y_i = 0|x_i > c\}]$$

Then, if

$$\#\{y_i = 0|x_i < \hat{c}\} + \#\{y_i = 1|x_i > \hat{c}\} \leq \#\{y_i = 1|x_i < \hat{c}\} + \#\{y_i = 0|x_i > \hat{c}\}$$

one will employ the classifier $\hat{f}(x) = I[x < \hat{c}]$ and otherwise the classifier $\hat{f}(x) = I[x > \hat{c}]$. Simulate 10,000 training samples and find corresponding classifiers \hat{f} . For each \hat{f} compute a (conditional on the training sample) error rate (an average of two appropriate normal probabilities on half infinite intervals bounded by \hat{c}) and average across the training samples. What is the "fitting penalty" for this procedure? Redo this exercise, using a training set of size $N = 50$. Is the fitting penalty larger than for $N = 100$?

20. (6HW-17) Consider the model of Problem 19 above, but change to the "-1 and 1" coding of classes/values of y .

a) Plot the function g minimizing $E\exp(-yg(x))$ over all choices of real-valued g .

Suppose then that one wishes to approximate this minimizer from part **a)** with a function of the form $\beta_0 + \beta_1(x - \bar{x}) + \beta_2(x - \bar{x})^2$ based on a training set. Your instructor will provide a training set of size $N = 100$ based on the model of this problem. Use it in what follows.

b) Use a numerical optimizing routine and identify values $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2$ minimizing the empirical average loss

$$R(\beta_0, \beta_1, \beta_2) = \frac{1}{N} \sum_{i=1}^N \exp\left(-y_i \left(\beta_0 + \beta_1(x_i - \bar{x}) + \beta_2(x_i - \bar{x})^2\right)\right)$$

c) Now consider the penalized fitting problem where one chooses to optimize

$$R_\lambda(\beta_0, \beta_1, \beta_2) = \frac{1}{N} \sum_{i=1}^N \exp\left(-y_i \left(\beta_0 + \beta_1(x_i - \bar{x}) + \beta_2(x_i - \bar{x})^2\right)\right) + \lambda\beta_2^2$$

For several different values of $\lambda > 0$, plot on the same set of axes, the optimizer from **a)**, the function $\beta_0 + \beta_1(x - \bar{x}) + \beta_2(x - \bar{x})^2$ optimizing $R(\beta_0, \beta_1, \beta_2)$ from **b)**, and the functions optimizing $R_\lambda(\beta_0, \beta_1, \beta_2)$.

21. (5E2-14) At a particular input vector of interest in a SEL prediction problem, say \mathbf{x} , the conditional mean of $y|\mathbf{x}$ is 3. Two different predictors, $\hat{f}_1(\mathbf{x})$ and $\hat{f}_2(\mathbf{x})$ have biases (across random selection of training sets of fixed size N) at this value of \mathbf{x} that are respectively .1 and $-.5$. The random vector of predictors at \mathbf{x} (randomness coming from training set selection) has covariance matrix

$$\text{Cov}\left(\begin{array}{c} \hat{f}_1(\mathbf{x}) \\ \hat{f}_2(\mathbf{x}) \end{array}\right) = \begin{pmatrix} 1 & .25 \\ .25 & 1 \end{pmatrix}$$

If one uses a linear combination of the two predictors

$$\hat{f}^{\text{ensemble}}(\mathbf{x}) = a\hat{f}_1(\mathbf{x}) + b\hat{f}_2(\mathbf{x})$$

there are optimal values of the constants a and b in terms of minimizing the expected (across random selection of training sets) squared difference between $\hat{f}^{\text{ensemble}}(\mathbf{x})$ and 3 (the conditional mean of $y|\mathbf{x}$). Write out and optimize an explicit function of a and b that (in theory) could be minimized in order to find these optimal constants.

22. (5E1-18) Consider a $p = 1$ SEL prediction problem where

$$E[y|x] = x(1-x), \text{Var}[y|x] = x(1-x), \text{ and } x \sim U(0,1)$$

- a) Find the expected loss of a theoretically optimal predictor of y , $f^{\text{opt}}(x)$.
- b) Consider predictors of the form

$$f_c(x) = c_1I[0 \leq x < .4] + c_2I[.4 \leq x < .6] + c_3I[.6 \leq x \leq 1]$$

for real constants $c_1, c_2,$ and c_3 . Find

$$E[y|0 \leq x < .4], E[y|.4 \leq x < .6], \text{ and } E[y|.6 \leq x \leq 1]$$

and argue that these give optimal values for the constants.

c) Give an explicit expression for the expected loss of the optimal predictor of the form $f_c(x)$. Note that together with the first answer this could give the modeling penalty here.

d) Give an explicit expression for the fitting penalty if based on a training set of size N , the value c_l is estimated by

$$\hat{c}_l = \bar{y}_l I[\text{at least one } x_i \text{ is in the interval corresponding to } c_l]$$

(where \bar{y}_l is the sample mean response for training cases with x_i in the interval corresponding to c_l).

23. (5HW-18) Consider a SEL prediction problem where $p = 1$, and the class of functions used for prediction is the set of linear functions

$$S = \{h|h(x) = b_0 + b_1x \forall x \text{ and some } b_0, b_1 \in \mathfrak{R}\}$$

Suppose that in fact

$$x \sim U(0, 1), E[y|x] = x + 2x^2, \text{ and } \text{Var}[y|x] = .25x^2$$

a) Under this model, what is the best element of S , say g^* , for predicting y ? Use this to find the modeling penalty/average squared model bias in this problem.

b) What is the smallest possible expected loss here (the mean squared prediction error of the theoretically best predictor, $f(x) = x + 2x^2$)?

Now consider the situation where $N = 50$ and simple linear regression (OLS) is used to choose an element of S based on a training set. Simulate a large number of training sets (at least 1000 of them) of this size according to the model here using normal conditional distributions for $y|x$. For each simulated training set, find the simple linear regression slope and intercept and use these to estimate the mean vector and covariance matrix for the fitted regression coefficients (for this sample size and this model). Use the estimated mean and covariance as follows.

c) Estimate the linear function of x that is the difference between your answer to **a)** and the average linear function produced by SLR in this context. Find the expected square of this difference according to the $U(0, 1)$ distribution of x . (This is an estimate of the expected squared fitting bias here.)

d) Using your estimated covariance matrix, approximate the function of x that is the variance (across training sets) of the value on the least squares line at x . Find the mean of this function according to the $U(0, 1)$ distribution of x . (This is an estimate of the expected prediction variance.)

e) In light of **c)** and **d)** what is the (estimated by simulation) fitting penalty in this context? What then is an approximate value for Err ?

24. (6HW-17) Consider the Ames house price dataset of Problem 5 above and the famous Wisconsin breast cancer dataset on the UCI Machine Learning Data Repository. The latter has $683 = 699 - 16$ complete cases (16 cases are incomplete) with $p = 9$ numerical characteristics of biopsied tumors, 239 of which were malignant and 444 which were benign. Use the `train()` function in the `caret` package in R and do the following.

a) Find a best k for k -nn SEL prediction of home selling price first using repeated 8-fold cross-validation, and then LOO cross-validation. Be sure to use standardized inputs (even for the 0-1 indicators) and to re-standardize for each fold. Plot the cross-validation root mean squared prediction error as a function of k . How does the training root mean squared prediction error for the best k compare to the corresponding cross-validation root mean squared prediction error?

b) Find a best k for k -nn classification between benign and malignant cases based on 0-1 loss, first using repeated 10-fold cross-validation, and then LOO cross-validation. Be sure to use standardized inputs and to re-standardize for each fold. Plot the cross-validation classification error rate as a function of k .

How does the training error rate for the best k compare to the corresponding cross-validation error rate?

25. (5E2-14) Below are class-conditional pmfs for a discrete predictor variable x in a $K = 3$ class 0-1 loss classification problem. Suppose that probabilities of $y = k$ for $k = 1, 2, 3$ are $\pi_1 = .4, \pi_2 = .3$, and $\pi_3 = .3$. For each value of x give the corresponding value of the optimal (Bayes) classifier f^{opt} .

$y \backslash x$	1	2	3	4	5	6
1	.2	.1	.2	.1	.1	.3
2	.1	.1	.3	.3	.1	.1
3	.2	.1	.2	.2	.2	.1

26. (5E2-14) A training set of size $N = 3000$ produces counts of (x, y) pairs as in the table below. (Assume these represent a random sample of all cases.) For each value of x give the corresponding value of an approximately optimal 0-1 loss (Bayes) classifier \hat{f} .

$y \backslash x$	1	2	3	4	5	6
1	95	155	145	205	105	150
2	305	105	195	140	195	155
3	150	190	160	155	150	245

27. (5E1-20) In a $p = 1$ SEL prediction problem, suppose that (unknown to a statistical learner) $x \sim U(0, 1)$ and $E[y|x] = x^3$.

a) Two sets of functions mapping $(0, 1) \rightarrow \mathfrak{R}$, $S_1 = \{c\}_{c \in \mathfrak{R}}$ and $S_2 = \{dx\}_{d \in \mathfrak{R}}$, might be searched for a suitable function to predict target y based on input x . Determine which set provides the smaller model bias.

b) Suppose that (again unknown to the statistical learner) $\text{Var}[y|x] = x^2$ from which it follows that $Ey = \frac{1}{4}$ and $\text{Var}y = \frac{139}{336}$. The statistical learner uses $S_1 = \{c\}_{c \in \mathfrak{R}}$ and the predictor $\hat{f}(x) = \bar{y}_N$ (the sample mean output from N training cases). What then is the value of the fitting penalty?

28. (5E1-20) Consider a $p = 1$ SEL prediction problem. Suppose that a predictor of the form

$$\hat{f}_c(x) = c \cdot \left(\frac{\sum_{i=1}^N x_i y_i}{\sum_{i=1}^N x_i^2} \right) \cdot x$$

is to be used. (This is a multiple of the least squares slope estimate in a no-intercept regression model based on the training set.) The multiplier, $c > 0$ remains to be chosen. Suppose that training data are as follows.

x	-3	-1	0	0	4
y	-2	-1	0	1	2

a) Write out an explicit form for the leave-one-out-cross-validation-mean-squared-prediction-error for $\hat{f}_c(x)$ in this toy example. (This is a function of the real variable c , say $CV(c)$.)

b) The value of c minimizing $CV(c)$ in **a)** turns out to be $\hat{c} = .9784$. Show this. Why is $CV(.9784)$ **not** a good indicator of the effectiveness of prediction methodology that in general employs form \hat{f}_c with c chosen by optimizing $CV(c)$? How would you produce a reliable predictor of the performance of $\hat{f}_{\hat{c}}$ in this problem? (Explain clearly and completely.)

29. (5E1-20) In a $K = 3$ class classification problem with $p = 1$, class conditional pdfs for x on $(0, 1)$ are

$$p(x|1) = I[0 < x < 1], p(x|2) = 3x^2 I[0 < x < 1],$$

$$\text{and } p(x|3) = 3(1-x)^2 I[0 < x < 1]$$

a) With class probabilities $\pi_1 = \pi_2 = \pi_3 = \frac{1}{3}$ and 0-1 loss, give an explicit form of a theoretically optimal classifier $f(x)$ and evaluate the minimum possible overall error rate (the expected loss of your optimal classifier).

b) With class probabilities $\pi_1 = .25, \pi_2 = .375$, and $\pi_3 = .375$ and 0-1 loss, give an explicit form of a theoretically optimal classifier $f(x)$ and evaluate the corresponding class-conditional error rates (conditional probabilities of a misclassification for $y = 1, y = 2$, and $y = 3$).

30. Suppose that in a $p = 1$ context, one is to predict y under squared error loss on the basis of a training set $\mathbf{T} = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$, and for

$$r_i = \frac{y_i}{x_i} \quad \text{and} \quad \bar{r} = \frac{1}{N} \sum_{i=1}^N r_i$$

under consideration are the two very simple predictors

$$\hat{f}_1(x) = x \quad \text{and} \quad \hat{f}_2(x) = \bar{r} \cdot x$$

Under the usual setup where the N pairs in \mathbf{T} are iid according to P independent of $(x, y) \sim P$, consider P defined by a marginal distribution $x \sim U(\frac{1}{2}, \frac{3}{2})$ and conditional distributions $y|x \sim N(\beta x, \sigma^2)$.

a) Show that

$$\text{Err}_1 = E\left(y - \hat{f}_1(x)\right)^2 = \sigma^2 + \frac{13}{12}(\beta - 1)^2$$

and that

$$\text{Err}_2 = E\left(y - \hat{f}_2(x)\right)^2 = \sigma^2 + \frac{13}{9N}\sigma^2$$

so that the first predictor is preferable to the second provided $(\beta - 1)^2/4 < \sigma^2/3N$ i.e. provided $(\beta - 1)^2/\sigma^2 < 4/3N$ (a fact that is of no practical use to a statistical learner not in full possession of the model generating the training set!).

Consider LOOCV-guided choice between the two simple predictors for this problem. The LOOCVMSPE for $\hat{f}_1(x)$ is

$$CV_1 = \frac{1}{N} \sum_{i=1}^N (y_i - x_i)^2 = \frac{1}{N} \sum_{i=1}^N y_i^2 - 2 \frac{1}{N} \sum_{i=1}^N y_i x_i + \frac{1}{N} \sum_{i=1}^N x_i^2$$

Then for $\bar{r}_{(i)} = \frac{1}{N-1} \sum_{j \neq i, j=1}^N r_j$, the LOOCVMSPE for $\hat{f}_2(x)$ is

$$CV_2 = \frac{1}{N} \sum_{i=1}^N (y_i - \bar{r}_{(i)} x_i)^2 = \frac{1}{N} \sum_{i=1}^N y_i^2 - 2 \frac{1}{N} \sum_{i=1}^N \bar{r}_{(i)} y_i x_i + \frac{1}{N} \sum_{i=1}^N \bar{r}_{(i)}^2 x_i^2$$

So $CV_1 < CV_2$ when

$$2 \sum_{i=1}^N \bar{r}_{(i)} y_i x_i - 2 \sum_{i=1}^N y_i x_i < \sum_{i=1}^N \bar{r}_{(i)}^2 x_i^2 - \sum_{i=1}^N x_i^2$$

that is, when

$$2 \sum_{i=1}^N (\bar{r}_{(i)} - 1) y_i x_i < \sum_{i=1}^N (\bar{r}_{(i)}^2 - 1) x_i^2$$

Thus, the pick-the(-cross-validation-)winner predictor here is

$$\begin{aligned} \tilde{f}(x) = xI & \left[2 \sum_{i=1}^N (\bar{r}_{(i)} - 1) y_i x_i < \sum_{i=1}^N (\bar{r}_{(i)}^2 - 1) x_i^2 \right] \\ & + \bar{r} xI \left[2 \sum_{i=1}^N (\bar{r}_{(i)} - 1) y_i x_i \geq \sum_{i=1}^N (\bar{r}_{(i)}^2 - 1) x_i^2 \right] \end{aligned}$$

The generalization/prediction error for this pick-the-winner predictor is

$$\text{Err}_{\text{ptw}} = \text{E} \left(y - \tilde{f}(x) \right)^2$$

which is certainly NOT just $\min(\text{Err}_1, \text{Err}_2)$. Further, this prediction error Err_{ptw} is NOT naively approximated by the cross-validation error of the winner

$$\min(CV_1, CV_2)$$

b) To demonstrate all this, generate 1000 simulated training sets of size $N = 27$ and an additional observation pair (x, y) for each of these, using $\sigma = 2$ for values of $\beta = \frac{3}{9}, \frac{5}{9}, \frac{7}{9}, \dots, \frac{15}{9}$. (This is 7 sets—one for each β considered—of 1000 training sets, each of size $N = 27$.) For each training set, find $\tilde{f}(x)$ and $(y - \tilde{f}(x))^2$ and average the squared differences across the 1000 sets in each group to produce a simulation-based estimate of Err_{ptw} for each value of β . How do these averages compare to the values of $\min(\text{Err}_1, \text{Err}_2)$ for these cases? For each value of β compare the distribution of 1000 random values $\min(CV_1, CV_2)$ produced, to the approximate value of Err_{ptw} . Does the random variable $\min(CV_1, CV_2)$ appear to be a good estimator of Err_{ptw} ? Does it appear to be biased, and if so, in what direction?

c) Should one wish to make an honest empirical assessment of the likely performance of $\tilde{f}(x)$, what can be done using LOOCV is this. For each "fold" consisting of case i use the "remainder" consisting of the other $N - 1$ cases to compute a "remainder i version" of the pick-the-winner predictor, say $\tilde{f}^{(i)}$.

That is, let $\bar{r}_{(i,j)} = \frac{1}{N-2} \sum_{l \neq i, l \neq j, l=1}^N r_l$ and define

$$\begin{aligned} \tilde{f}^{(i)}(x) = xI & \left[2 \sum_{j \neq i, j=1}^N (\bar{r}_{(i,j)} - 1) y_j x_j < \sum_{j \neq i, j=1}^N (\bar{r}_{(i,j)}^2 - 1) x_j^2 \right] \\ & + \bar{r}_{(i)} xI \left[2 \sum_{j \neq i, j=1}^N (\bar{r}_{(i,j)} - 1) y_j x_j \geq \sum_{j \neq i, j=1}^N (\bar{r}_{(i,j)}^2 - 1) x_j^2 \right] \end{aligned}$$

and use $\tilde{f}^{(i)}(x_i)$ in predicting y_i . The appropriate LOOCV error is then

$$CV_{\text{ptw}} = \frac{1}{N} \sum_{i=1}^N (y_i - \tilde{f}^{(i)})^2.$$

For the case of β in part **b)** with the worst match between Err_{ptw} and the distribution of the variable $\min(CV_1, CV_2)$, find the 1000 values of CV_{ptw} . Does the random variable CV_{ptw} seem to be a better estimator of Err_{ptw} than the naive $\min(CV_1, CV_2)$? Explain.

31. Consider the case of random variables $C(i)$ for $i \in \mathcal{I}$ (some index set) and let \mathbf{C} stand for the random vector/function with coordinates/entries $C(i)$. Define the random variable

$$i^* = \arg \min_{i \in \mathcal{I}} C(i)$$

(a minimizer of the entries of \mathbf{C}). (We'll assume enough regularity here that there are no issues in defining this variable or any of the probabilities or expectations used here.)

Suppose that of interest is the (non-random) vector/function $\mathbf{E}\mathbf{C}$, its (non-random) optimizer

$$i^{\text{opt}} = \arg \min_{i \in \mathcal{I}} (\mathbf{E}C(i))$$

and its minimum/optimum value $EC(i^{\text{opt}})$.

a) Why is it "obvious" that

$$EC(i^*) \leq EC(i^{\text{opt}}) \quad ?$$

b) Argue carefully that unless with probability 1 the non-random value i^{opt} is a minimizer of the random vector/function \mathbf{C} ,

$$EC(i^*) < EC(i^{\text{opt}})$$

c) Say what the line of thinking in this problem implies about cross-validation and a "pick-the-winner" prediction strategy. (Does it address the fact that almost always in predictive analytics contests, when final results based on prediction for new cases are revealed they are worse than what contestants expect for a test error?)

A.3 Section 1.4 Exercises

1. (5HW-16) Consider a 0-1 loss $K = 2$ classification problem with $p = 1$, $\pi_0 = \pi_1 = \frac{1}{2}$, and pdfs

$$p(x|0) = I[-.5 < x < .5] \quad \text{and} \quad p(x|1) = 12x^2 I[-.5 < x < .5]$$

a) What is the optimal classification rule in this problem?

b) If one were to do "feature engineering" here, adding some function of x , say $t(x)$, to make a vector of features $(x, t(x))$ for classification purposes (hoping to eventually employ a good "linear classifier"

$$\hat{f}(x, t(x)) = I[a + bx + ct(x) > 0]$$

for appropriate constants $a, b,$ and c), what (knowing the answer to **a**) would be a good choice of $t(x)$? (Of course, one doesn't know the answer to **a**) when doing feature selection!)

c) What is the "minimum expected loss possible" part of Err in this problem?

d) Identify the best classification rule of the form $g_c(x) = I[x > c]$. (This is $g^*(x)$ for $S = \{g_c\}$. This could be thought of as the 1-d version of a "best linear classification rule" here ... where linear classification is not so smart.) What is the "modeling penalty" part of Err in this situation?

e) Suggest a way that you might try to choose a classification rule g_c based on a very large training sample of size N . Notice that a large training set would allow you to estimate cumulative conditional probabilities $P[x \leq c|y]$ by relative frequencies

$$\frac{\# \text{ number of training cases with } x_i \leq c \text{ and } y_i = y}{\# \text{ number of training cases with } y_i = y}$$

2. (5E1-15) Consider two probability densities on the unit disk in \mathfrak{R}^2 (i.e. on $\{(x_1, x_2) \mid x_1^2 + x_2^2 \leq 1\}$),

$$p(x_1, x_2|1) = \frac{1}{\pi} \quad \text{and} \quad p(x_1, x_2|2) = \frac{3}{2\pi} \sqrt{1 - (x_1^2 + x_2^2)}$$

and a 2-class 0-1 loss classification problem with class probabilities $\pi_1 = \pi_2 = .5$.

a) Give a formula for a best-possible single feature $T(x_1, x_2)$.

b) Give an explicit form for the theoretically optimal classifier in this problem.

3. (5E1-18) Consider a $K = 3$ classification model with $p = 3$ class-conditional densities on $[0, 1]^3$

$$p(x_1, x_2, x_3|1) = 2x_1, p(x_1, x_2, x_3|2) = 2x_2, \quad \text{and} \quad p(x_1, x_2, x_3|3) = 2x_3$$

a) Identify two real-valued features $T_1(\mathbf{x})$ and $T_2(\mathbf{x})$ that together provide complete summarizations of all information about the class label $y \in \{1, 2, 3\}$ provided by $\mathbf{x} = (x_1, x_2, x_3)$.

b) For the case of $\pi_1 = \pi_2 = \pi_3 = \frac{1}{3}$ give the form of an optimal 0-1 loss classifier in terms of the values t_1 and t_2 of $T_1(\mathbf{x})$ and $T_2(\mathbf{x})$.

c) For the case of $\pi_1 = .6, \pi_2 = .4$, and $\pi_3 = 0$ where $L(\hat{y}, 1) = 10I[\hat{y} \neq 1]$ and otherwise $L(\hat{y}, y) = I[\hat{y} \neq y]$, give the form of an optimal classifier in terms of the value of $\mathbf{x} = (x_1, x_2, x_3)$.

4. (6E2-15) One can consider the possibility of "kernelizing" nearest-neighbor prediction. ("Kernelization" amounts to mapping $\mathbf{x} \in \mathfrak{R}^p$ to $\mathcal{K}(\mathbf{x}, \cdot)$ in the abstract function space, \mathcal{A} , and using inner products in that space—and corresponding distances—based on the kernel.) Using the Gaussian kernel $\mathcal{K}(\mathbf{x}, \mathbf{z}) = \exp(-\|\mathbf{x} - \mathbf{z}\|^2)$, what is the abstract-space distance between $\mathcal{K}(\mathbf{x}, \cdot)$ and $\mathcal{K}(\mathbf{z}, \cdot)$? Describe the set of training cases $\mathbf{x}_i \in \mathfrak{R}^p$ with $\mathcal{K}(\mathbf{x}_i, \cdot)$ in the k -nearest neighborhood of $\mathcal{K}(\mathbf{x}, \cdot)$ in the abstract space \mathcal{A} .

5. (6E1-17) In Section 1.4.3 there is an assertion that for a finite set B , say $B = \{b_1, b_2, \dots, b_m\}$, for $|A|$ the number of elements in $A \subset B$, one kernel function on subsets of B is

$$\mathcal{K}(A_1, A_2) = 2^{|A_1 \cap A_2|}$$

(B could, for example, be a list of attributes that an item might or might not possess.)

a) Prove that \mathcal{K} is a kernel function using the "kernel mechanics" facts. (Hint: You may find it useful to associate with each $A \subset B$ an m -dimensional vector of 0s and 1s, call it $\mathbf{x}_A \in \{0, 1\}^m$, with $x_{Ai} = 1$ exactly when $b_i \in A$.)

b) Let $T(A)(\cdot) = \mathcal{K}(A, \cdot) = 2^{|A \cap \cdot|}$ map subsets of B to real-valued functions of subsets of B . In the abstract space \mathcal{A} (of real-valued functions of subsets of B) what is the distance between $T(A)$ and $T(B)$, $\|T(A) - T(B)\|_{\mathcal{A}}$?

For N training "vectors" (A_i, y_i) ($A_i \subset B$ and $y_i \in \mathfrak{R}$) consider the corresponding N points in $\mathcal{A} \times \mathfrak{R}$, namely $(T(A_i), y_i)$ for $i = 1, \dots, N$. Define a k -neighborhood $N_k(V)$ of a point (function) $V \in \mathcal{A}$ to be a set of k points (functions) $T(A_i)$ with smallest $\|T(A_i) - V\|_{\mathcal{A}}$.

c) Carefully describe a SEL k -nn predictor of y , $f(V)$, mapping elements V of \mathcal{A} to real numbers \hat{y} in \mathfrak{R} . Then describe as completely as possible the corresponding predictor $f(T(A))$ mapping $A \subset B$ to $\hat{y} \in \mathfrak{R}$.

d) A more direct method of producing a kind of k -nn predictor of y is to take account of the hint for part **a)** and for subsets A and C of B , to associate m -vectors of 0s and 1s respectively \mathbf{x}_A and \mathbf{x}_C and define a distance between sets A and C as the Euclidean distance between \mathbf{x}_A and \mathbf{x}_C . This typically produces a different predictor than the one in part **c)**. Argue this point by considering distances from \mathbf{x}_A and \mathbf{x}_C and from \mathbf{x}_A and \mathbf{x}_D in \mathfrak{R}^m and from $T(A)$ and $T(C)$ and from $T(A)$ and $T(D)$ in the space \mathcal{A} for cases with $|A| = 10, |C| = 4, |D| = 5, |A \cap C| = 2$, and $|A \cap D| = 3$.

6. (6HW-13) For a $\gamma > 0$, consider the function $\mathcal{K} : \mathfrak{R}^2 \times \mathfrak{R}^2 \rightarrow \mathfrak{R}$ defined by

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = \exp\left(-\gamma \|\mathbf{x} - \mathbf{z}\|^2\right)$$

a) Use the facts about kernel functions in Section 1.4.3 to argue that \mathcal{K} is a kernel function. (Note that $\|\mathbf{x} - \mathbf{z}\|^2 = \langle \mathbf{x}, \mathbf{x} \rangle + \langle \mathbf{z}, \mathbf{z} \rangle - 2\langle \mathbf{x}, \mathbf{z} \rangle$.)

b) Argue that there is a $\varphi : \mathfrak{R}^2 \rightarrow \mathfrak{R}^\infty$ so that with (infinite-dimensional) feature vector $\varphi(\mathbf{x})$ the kernel function is a "regular \mathfrak{R}^∞ inner product"

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = \langle \varphi(\mathbf{x}), \varphi(\mathbf{z}) \rangle_\infty = \sum_{l=1}^{\infty} \varphi_l(\mathbf{x}) \varphi_l(\mathbf{z})$$

(You will want to consider the Taylor series expansion of the exponential function about $\mathbf{0}$ and coordinate functions of φ that are multiples of all possible products of the form $x_1^p x_2^q$ for non-negative integers p and q . It is not necessary to find explicit forms for the multipliers, though that can probably be done. You do need to argue carefully though, that such a representation is possible.)

7. (6E1-19) Consider a 3-class classification problem with input vector $\mathbf{x} \in [0, 1]^5$. Suppose that class-conditional densities for \mathbf{x} are of the forms

$$p(\mathbf{x}|1) = \frac{4}{3}(x_1 x_2 + x_4), p(\mathbf{x}|2) = \frac{4}{3}(x_3 x_4 + x_5), \text{ and } p(\mathbf{x}|3) = 4x_4 x_5$$

(all on $[0, 1]^5$) and that class probabilities are $\pi_1 = \pi_2 = \pi_3 = \frac{1}{3}$.

a) Give expressions for the smallest set of features possible for representing \mathbf{x} without loss of information in this model.

b) Evaluate the conditional probability that $y = 1$ given that $\mathbf{x} = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$.

c) Give explicit prescriptions for conditions on \mathbf{x} under which an optimal classifier (say, $f(\mathbf{x})$) has $f(\mathbf{x}) = 1$, under which $f(\mathbf{x}) = 2$, and under which $f(\mathbf{x}) = 3$.

8. (6E1-19) "Correlation functions" from time series and spatial modeling (and analysis of "computer experiments") are a source of reproducing kernels for use in machine learning. In a 1992 paper, Mitchell and Morris introduced the useful correlation function

$$\rho(d) = \begin{cases} 1 - 6d^2 + 6|d|^3 & \text{if } |d| < .5 \\ 2(1 - |d|)^3 & \text{if } .5 \leq |d| \leq 1 \\ 0 & \text{if } |d| > 1 \end{cases}$$

(Interestingly, $\rho(d)$ is a natural cubic spline.⁵⁵) Here we will use it to make the reproducing kernel

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) \equiv \rho(\|\mathbf{x} - \mathbf{z}\|)$$

mapping $\mathfrak{R}^p \times \mathfrak{R}^p \rightarrow \mathfrak{R}$. For sake of concreteness, take $p = 2$.

a) For the mapping from \mathfrak{R}^2 to the abstract function space \mathcal{A} defined by the kernel $T(\mathbf{x})(\cdot) \equiv \mathcal{K}(\mathbf{x}, \cdot)$, find numerical values for

- $\|T(\mathbf{x})\|_{\mathcal{A}}$
- $\langle T((0, 0)) + 2T((\frac{1}{2}, \frac{1}{2})), 3T((1, 1)) \rangle_{\mathcal{A}}$
- $\|T((0, 0)) - T((0, .6))\|_{\mathcal{A}}$

b) Consider the problem of (penalized SEL) prediction of y from $\mathbf{x} = (x_1, x_2)$ based on N training cases. Suppose that one will "correct" ridge regression by addition of an appropriate linear combination of the functions $\mathcal{K}(\mathbf{x}_i, \cdot)$ to produce a final predictor. That is, for centered y s and standardized x s consider a predictor of form

$$f(\mathbf{x}) = \alpha_1 x_1 + \alpha_2 x_2 + \sum_{i=1}^N \beta_i \mathcal{K}(\mathbf{x}_i, \mathbf{x})$$

using for $\lambda_1 > 0$ and $\lambda_2 > 0$ a penalty

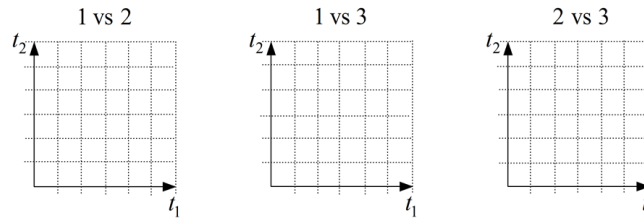
$$\lambda_1 (\alpha_1^2 + \alpha_2^2) + \lambda_2 \left\| \sum_{i=1}^N \beta_i \mathcal{K}(\mathbf{x}_i, \cdot) \right\|_{\mathcal{A}}^2$$

(that penalizes both the "size" of the linear part of the predictor and the "size" of the kernel-based correction to it). Develop (for fixed λ_1 and λ_2 and training set and using notation \mathbf{K} for the Gram matrix) a quadratic function of the coefficients $\alpha_1, \alpha_2, \beta_1, \beta_2, \dots, \beta_N$ that you would optimize to produce a predictor.

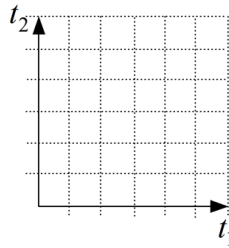
9. (6E2-15) A 3-class classification model has $\pi_k = P[y = k] = \frac{1}{3}$ for $k = 1, 2, 3$, and densities $p(\mathbf{x}|k)$ for the conditional distributions of $\mathbf{x}|y = k$, $k = 1, 2, 3$. For some pair of features $T_1(\mathbf{x})$ and $T_2(\mathbf{x})$ show that:

⁵⁵See Section 4.2 for the meaning of this language.

a) Optimal classification for each of the 3 *pairs* of classes is linear classification based on the features t_1 and t_2 . (Define the features and show the linear classification boundaries on axes like those below. Indicate the scales for the features.)



b) The optimal 3-class classifier can be realized as a "OVO" (one-versus-one) combination of the three 2-class classifiers. (Show the optimal classification boundaries in terms of features t_1 and t_2 and indicate which regions correspond to which classification decisions.)



10. (5HW-16) Return to the context of Problem 13 of Section A.2 and the last/largest set of predictors. Center the y vector to produce (say) \mathbf{Y}^* , remove the column of 1s from the \mathbf{X} matrix (giving a 100×9 matrix) and standardize the columns of the resulting matrix, to produce (say) \mathbf{X}^* .

a) If one somehow produces a coefficient vector β^* for the centered and standardized version of the problem, so that

$$\hat{y}^* = \beta_1^* x_1^* + \beta_2^* x_2^* + \dots + \beta_9^* x_9^*$$

what is the corresponding predictor for y in terms of

$$\{1, x, x^2, x^3, x^4, x^5, \sin x, \cos x, \sin 2x, \cos 2x\} \quad ?$$

b) Do the transformations and fit the equation in a) by OLS. How do the fitted coefficients and error sum of squares obtained here compare to what you get simply doing OLS using the raw data (and a model including a constant term)?

11. (5HW-18) Consider a toy 3-class classification problem with conditional distributions $x|y$ that are $N(0, 1)$ for $y = 1$, $N(1, (.5)^2)$ (the standard deviation is .5) for $y = 2$, and $N(2, 1)$ for $y = 3$ and class probabilities that are $\pi_1 = \pi_2 = \pi_3 = 1/3$.

a) Plot the three functions

$$P[y = 1|x], P[y = 2|x], \text{ and } P[y = 3|x]$$

b) The exposition identifies an optimal pair of "features" for this 3-class problem. Plot those two features, say $t_1(x)$ and $t_2(x)$ on the same set of axes.

c) Show that the optimal 3-class 0-1 loss classifier for any set of class probabilities π_1, π_2 , and π_3 can be written as a function of the features from b).

12. (5E1-20) 4. It is well-known that $\mathcal{K}(z, x) = (1 + zx)^2$ mapping $\Re^2 \rightarrow \Re$ is a legitimate "kernel function."

a) Suppose that for the training data of Problem 28 in Section A.2, one determines to fit a predictor for y of the form

$$\hat{f}(x) = \sum_{i=1}^5 \beta_i \mathcal{K}(x, x_i)$$

by penalized least squares, using a (λ) multiple of the abstract (reproducing-kernel-function-space) squared norm of \hat{f} as the penalty. Write out in completely explicit form the quantity to be minimized in order to do the fitting. (This is a function of $\beta_1, \beta_2, \dots, \beta_5$ and λ . You don't need to do scalar or matrix algebraic simplification, but your answer must evaluate to a number when values for the β s and λ are plugged in.)

b) $T(x)(\cdot) = \mathcal{K}(x, \cdot)$ is non-linear map $\Re \rightarrow A$. Show that the span of $\{T(x_1), \dots, T(x_5)\}$ in A is 3-dimensional. (What kinds of functions of a single real variable are mapped onto by T ?) In light of this fact and the nature of the functions $T(x_i)(\cdot)$ propose a different penalized least squares fitting problem that has the same set of possible predictors as in a) but requires optimization over only 3 coefficients $\alpha_1, \alpha_2, \alpha_3$ (for a given penalty weight " λ "). (You do not need to try to match the objective in a) exactly. You need only to provide a sensible penalized version of fitting over the same set of functions.)

13. (6E2-13) Below are 3 (of hypothetically many) text "documents" in a corpus using the alphabet $\mathbb{A} = \{a, b\}$. Consider preparing a data matrix for text processing for such documents. In particular, for each of the documents below, prepare a row of a data matrix consisting of all 1-gram frequencies, all 2-gram frequencies, and a feature quantifying the discounted (use $\lambda = .5$) appearances of the interesting string "aaaa" in the documents. (In computing this latter feature, count only strings with exactly 4 a's in them. Don't, for example, count strings with 5 a's by ignoring one of the interior a's.)

Document 1: a a b a b b a a a b b b b a a a b a b a

Document 2: a a a b b b a b a a

Document 3: b b b b a b a b b a

A.4 Section 1.5 Exercises

1. (6E1-17) Consider the 2-class classification model with the coding $y \in \{-1, 1\}$ and (for sake of concreteness) $x \in \mathfrak{R}^1$. For $g(x)$ a generic voting function we'll consider the classifier

$$f(x) = \text{sign}(g(x))$$

Another (besides those mentioned in the exposition) "function loss" sometimes discussed is

$$h(v) = (v - 1)^2$$

a) Carefully derive the function $g^{\text{opt}}(x)$ optimizing $Eh(yg(x))$ over choices of g .

b) To the extent possible, simplify a good upper bound on the 0-1 loss error rate of a classifier $f(x)$ made from your $g^{\text{opt}}(x)$ from part **a**).

c) Suppose that in pursuit of a good classifier, one wishes to optimize an empirical version of $Eh(yg(x))$, based on a training set of size N , over the class of functions of the form

$$g(x|\beta_0, \beta_1) = 2\Phi(\beta_0 + \beta_1 x) - 1$$

penalized by $\lambda\beta_1^2$ for a $\lambda \geq 0$. (Φ is the standard normal cdf.) In as simple a form as possible, give two equations to be solved simultaneously to do this fitting.

d) Suppose that as a matter of fact the two class-conditional densities operating are

$$p(x|-1) = I[0 < x < 1] \quad \text{and} \quad p(x|1) = 6x(1-x)I[0 < x < 1]$$

and that ultimately what is desired is a good ordering function $\mathcal{O}(x)$, one that produces a small value of the "AUC" criterion. Do you expect the methodology of part **c**) to produce a function $g(x|\hat{\beta}_0, \hat{\beta}_1)$ that would be a good choice of $\mathcal{O}(x)$? Explain carefully.

2. (6HW-17) Argue carefully that losses h_1, h_2 , and h_3 (negative Bernoulli loglikelihood term, exponential, and hinge losses) have optimizers of

$$Eh(yg(\mathbf{x}))$$

(functions $g^{\text{opt}}(\mathbf{x})$) as indicated in the exposition.

3. (5HW-18) In a 2-class classification problem using coding $\{-1, 1\}$ for the classes, the fake data below constitute a very small/toy training set.

y	-1	-1	1	1	1	-1	-1	1
x	1	2	3	4	5	6	7	8

Consider the production of a "voting function" of the form

$$g_{\mathbf{b}}(x) = \sum_{i=1}^8 b_i \exp(-c|x - x_i|^2)$$

by choice of the 8 coefficients b_i (for some choice of $c > 0$) under the "function loss" $h_2(u) = \exp(-u)$. (In the parlance of machine learning, the component functions $\exp(-c|x - x_i|^2)$ are data-dependent $p = 1$ "radial basis functions.") In fact, consider "penalized" fitting.

a) One possible penalized fitting criterion is

$$\frac{1}{8} \sum_{i=1}^8 \exp(-y_i g_{\mathbf{b}}(x_i)) + \lambda \sum_{i=1}^8 b_i^2$$

for some $\lambda > 0$. For choices of $c = .5$ and $c = 1$ optimize this criterion for two different values of $\lambda > 0$ and plot the four resulting voting functions on the same set of axes. Choose (by trial and error) two values of λ that produce clearly different optimizing functions. (`optim` in R or some other canned routine will be adequate to do this 8-d optimization.)

b) The function $\mathcal{K}(x, z) = \exp(-c|x - z|^2)$ is a "kernel function" in the sense of Section 1.4.3. That implies that the 8×8 Gram matrix

$$\mathbf{K} = (\mathcal{K}(x_i, x_j))_{\substack{i=1,2,\dots,8 \\ j=1,2,\dots,8}}$$

is non-negative definite. Thus, with $\mathbf{b} = (b_1, b_2, \dots, b_8)'$, $\mathbf{b}'\mathbf{K}\mathbf{b} \geq 0$ and another possible penalized fitting criterion replaces $\sum_{i=1}^8 b_i^2$ in part a) with $\mathbf{b}'\mathbf{K}\mathbf{b}$. For the same values of c and λ you used in part a) redo the optimization using this second penalization criterion and plot the resulting voting functions. Notice, by the way, that the penalty in a) is a $c \rightarrow \infty$ limit of this second penalty!

c) As indicated in Section 1.4.3, the mapping $T(x) = \mathcal{K}(x, \cdot)$ from \mathfrak{R}^1 to functions $\mathfrak{R}^1 \rightarrow \mathfrak{R}^1$ picks out $N = 8$ functions that are essentially normal pdfs. Linear combinations of these form a linear subspace of this function space. Further, there is a valid inner product that can be defined on this subspace, for which

$$\langle T(x), T(z) \rangle_{\mathcal{A}} = \mathcal{K}(x, z)$$

Using this inner product,

- what is the inner product of two elements of this subspace, say $g_{\mathbf{b}^*}(x)$ and $g_{\mathbf{b}^{**}}(x)$?
- what is the distance between $T(x)$ and $T(z)$,

$$\|T(x) - T(z)\|_{\mathcal{A}} = \langle T(x) - T(z), T(x) - T(z) \rangle_{\mathcal{A}}^{1/2} \quad ?$$

- how is the penalty in **b)** related to $\|g_{\mathbf{b}}\|_{\mathcal{A}}$ (the norm of the linear combination of functions in the function space)?

4. (6E2-13) Consider a toy 2-class classification problem with $p = 1$ and discrete conditional distributions of x indicated in the following table.

x	1	2	3	4	5	6	7	8	9	10
$p(x y = 1)$.04	.07	.06	.03	.24	0	.02	.09	.25	.2
$p(x y = 0)$.1	.1	.1	.1	.1	.1	.1	.1	.1	.1

a) If $P[y = 1] = 2/3$ what is the optimal classifier here and what is its error rate (for 0-1 loss)?

b) If one cannot observe x completely, but only

$$x^* = \begin{cases} 2 & \text{if } x \text{ is 1 or 2} \\ 4 & \text{if } x \text{ is 3 or 4} \\ 6 & \text{if } x \text{ is 5 or 6} \\ 8 & \text{if } x \text{ is 7 or 8} \\ 10 & \text{if } x \text{ is 9 or 10} \end{cases}$$

instead, what is the optimal classifier and what is its error rate (again assuming that $P[y = 1] = 2/3$ and using 0-1 loss)?

5. (6E1-19) Return to the situation of Problem 9 of Section A.2. For this toy dataset the 2 classes are balanced, and a 3-nearest-neighbor neighborhood has a fraction of "class 1" cases 0, $\frac{1}{3}$, $\frac{2}{3}$, or 1. Suppose that 3-nn results from this training set will be used to produce a 0-1 loss classifier for a scenario in which (there is severe class imbalance and) the actual probabilities of classes are $\pi_0 = .1$ and $\pi_1 = .9$. Find (and carefully argue that it is correct) the classification appropriate for an \mathbf{x} for which the 3-nearest-neighbor neighborhood has fraction $\frac{1}{3}$ of "class 1" cases.

6. (6E1-19) For voting function $g(\mathbf{x})$ in a 2-class classification problem (with $-1-1$ coding) and function losses h_1 and h_2 with $I[v < 0] \leq h_1(v) \leq h_2(v)$, presuming that $P_{-1}(g(\mathbf{x}) = 0|y = -1) = P_1(g(\mathbf{x}) = 0|y = 1) = 0$, the 0-1 loss error rate of the classifier $f(\mathbf{x}) = \text{sign}(g(\mathbf{x}))$, namely

$$\text{Err} = EI[yg(\mathbf{x}) < 0]$$

has upper bounds

$$b_1(g) = Eh_1(yg(\mathbf{x})) \quad \text{and} \quad b_2(g) = Eh_2(yg(\mathbf{x}))$$

a) Why do you know that $b_1(g) \leq b_2(g)$? Under what circumstances will it be the case that $b_1(g) < b_2(g)$?

b) If 1) g^* minimizes $b_1(g)$ over choices of g , 2) g^{**} minimizes $b_2(g)$ over choices of g , and 3) in fact your conditions in **a)** are met to imply that $b_1(g^{**}) <$

$b_2(g^{**})$, does it necessarily follow that g^* is a strictly better voting function (produces a better error rate) than g^{**} for the original 0-1 loss classification problem? Explain why or why not.

7. (DMC-19) The 2019 Data Mining Cup sponsored by Prudsys AG featured a classification problem for fraud detection based on numerical characteristics of self-checkout transactions at a retail location. The loss function employed, $L(\hat{y}, y)$ employed (actually, negative losses or "gains" were specified by the company), was for \hat{y} and y belonging to {fraud, no fraud}

$$L(\text{fraud, fraud}) = -5, L(\text{fraud, no fraud}) = 25, \\ L(\text{no fraud, fraud}) = 5, \text{ and } L(\text{no fraud, no fraud}) = 0$$

a) An optimal 2-class classifier for this problem decides in favor of fraud if $P[y = \text{fraud} | \mathbf{x}] > c$. Evaluate c .

b) An optimal 2-class classifier for this problem decides in favor of fraud if $\mathcal{L}(\mathbf{x}) \equiv (p(\mathbf{x} | \text{fraud}) / p(\mathbf{x} | \text{no fraud})) > c^*$ where c^* depends upon π_{fraud} . Evaluate c^* for $\pi_{\text{fraud}} = .1, .01, \text{ and } .001$.

8. (5HW-18) For the toy scenario of Problem 11 of Section A.3, consider a 2-class classification model for $y = 1$ and $y = 2$. Suppose the object is to produce a function $\mathcal{O}(x)$ minimizing (for independent $x \sim p(x|1)$ and $x^* \sim p(x|2)$)

$$P[\mathcal{O}(x) < \mathcal{O}(x^*)]$$

a) Plot an optimizing function.

b) Cases $i = 1, 2, \dots, 60$ in a hypothetical test set have $x_i = -2 + (i/10)$ and you must make an ordering of the test cases from "least to most likely" to have corresponding $y_i = 2$. Assign values 1 through 60 to the test set cases (1 \leftrightarrow least likely to 60 \leftrightarrow most likely) that you would submit in a predictive analytics contest where the "AUC criterion" is used to judge performance.

A.5 Section 1.6 Exercises

1. (5HW-14) Return to the context of Problem 7 Section A.2.

a) Find the marginal densities for all of the $p((x_1, x_2) | k)$. Define 4 new densities $p^*((x_1, x_2) | k)$ on the unit square by the products of the 2 marginals for the corresponding $p((x_1, x_2) | k)$. Consider a 0-1 loss $K = 4$ classification problem? approximating? the one in the original problem by using the $p^*((x_1, x_2) | k)$ in place of the $p((x_1, x_2) | k)$ for the $\boldsymbol{\pi} = (.25, .25, .25, .25)$ case. Make a 101×101 grid of points of the form $(i/100, j/100)$ for integers $0 \leq i, j \leq 100$ and for each such point determine the value of the optimal classifier for this new problem. Using these values, make a plot (using a different plotting color and/or symbol for each value of \hat{y}) showing the regions in $(0, 1)^2$ where the optimal classifier classifies to each class. Compare this plot to the one in Problem 7 of Section A.2. (The classifier here might be called a "naïve Bayes" classifier.)

b) Find the $p((x_1, x_2) | k)$ conditional densities for $x_2 | x_1$. Note that based on these and the marginals in part **a)** you can simulate pairs from any of the 4 joint distributions by first using the inverse probability transform of a uniform variable to simulate from the x_1 marginal and then using the inverse probability transform to simulate from the conditional of $x_2 | x_1$. (It's also easy to use a rejection algorithm based on (x_1, x_2) pairs uniform on $(0, 1)^2$.)

c) Generate 2 datasets consisting of multiple independent pairs (\mathbf{x}, y) where y is uniform on $\{1, 2, 3, 4\}$ and conditioned on $y = k$, the variable \mathbf{x} has density $p((x_1, x_2) | k)$. Make first a small training set with $N = 400$ pairs (to be used below). Then make a larger test set of 10,000 pairs. Use the test set to evaluate the (conditional on the training set) error rates of the optimal rule from Problem 7 Section A.2 and then the "naïve" rule from part **a)**.

d) Based on the $N = 400$ training set from **c)**, for several different numbers of neighbors (say 1, 3, 5, 10) make a plot like that required in part **c)** showing the regions where the nearest neighbor classifier classifies to each of the 4 classes. Then evaluate the (conditional on the small training sets) test error rates for the nearest neighbor rules.

e) Based on the training set, one can make estimates of the 2-d densities as

$$\hat{p}(\mathbf{x} | k) = \frac{1}{\#[i \text{ with } y_i = k]} \sum_{i \text{ with } y_i = k} h(\mathbf{x} | \mathbf{x}_i, \sigma^2)$$

for $h(\cdot | \boldsymbol{\mu}, \sigma^2)$ the bivariate normal density with mean vector $\boldsymbol{\mu}$ and covariance matrix $\sigma^2 \mathbf{I}$. (Try perhaps $\sigma \approx .1$.) Using these estimates and the relative frequencies of the possible values of y in the training set

$$\hat{\pi}_k = \frac{\#[i \text{ with } y_i = k]}{N}$$

an approximation of the optimal classifier is

$$\hat{f}(\mathbf{x}) = \arg \max_k \hat{\pi}_k \hat{p}(\mathbf{x} | k) = \arg \max_k \sum_{i \text{ with } y_i = k} h(\mathbf{x} | \mathbf{x}_i, \sigma^2)$$

Make a plot like that required in part **a)** showing the regions where this classifies to each of the 4 classes. Then evaluate the (conditional on the training set) test error rate for this classifier.

A.6 Section 2.1 Exercises

1. (5E1-16) Kernel methods in statistical learning are built on the fact that for a legitimate kernel function $\mathcal{K}(\mathbf{x}, \mathbf{z})$ there is an abstract linear space \mathcal{A} and a (non-linear) transform $T(\mathbf{x})$ from \mathfrak{R}^p to that space for which the inner product of transformed elements of \mathfrak{R}^p is

$$\langle T(\mathbf{x}), T(\mathbf{z}) \rangle_{\mathcal{A}} = \mathcal{K}(\mathbf{x}, \mathbf{z})$$

Use the Gaussian kernel function $\mathcal{K}(\mathbf{x}, \mathbf{z}) = \exp(-\|\mathbf{x} - \mathbf{z}\|^2)$ in what follows. ($\|\cdot\|$ is the usual \mathfrak{R}^p norm.)

a) For an input vector $\mathbf{x}_i \in \mathfrak{R}^2$, what is the norm of $T(\mathbf{x}_i)$ in the abstract space?

b) For input vectors $\mathbf{x}_i \in \mathfrak{R}^2$ and $\mathbf{x}_l \in \mathfrak{R}^2$, how is the distance between $T(\mathbf{x}_i)$ and $T(\mathbf{x}_l)$ in the abstract space related to the distance between \mathbf{x}_i and \mathbf{x}_l in \mathfrak{R}^p ?

2. (6E1-11) Consider the p -dimensional input space \mathfrak{R}^p and kernel functions mapping $\mathfrak{R}^p \times \mathfrak{R}^p \rightarrow \mathfrak{R}$.

a) Show that for $\phi : \mathfrak{R}^p \rightarrow \mathfrak{R}$, the function $\mathcal{K}(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})\phi(\mathbf{z})$ is a valid kernel. (You must show that for distinct $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$, the $N \times N$ matrix $\mathbf{K} = (\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j))$ is non-negative definite.)

b) Show that for two kernels $\mathcal{K}_1(\mathbf{x}, \mathbf{z})$ and $\mathcal{K}_2(\mathbf{x}, \mathbf{z})$ and two positive constants c_1 and c_2 , the function $c_1\mathcal{K}_1(\mathbf{x}, \mathbf{z}) + c_2\mathcal{K}_2(\mathbf{x}, \mathbf{z})$ is a kernel.

c) By virtue of **a)** and **b)**, the functions $\mathcal{K}_1(x, z) = 1 + xz$ and $\mathcal{K}_2(x, z) = 1 + 2xz$ are both kernels on $[-1, 1]^2$. They produce inner product spaces of functions. Show these are different.

3. (6E1-15) Consider the small space of functions on $[-1, 1]^2$ that are linear combinations of the 4 functions $1, x_1, x_2$, and x_1x_2 , with inner product defined by $\langle h, g \rangle = \iint_{[-1, 1]^2} h(x_1, x_2)g(x_1, x_2)dx_1dx_2$. Find the element of this space

closest to $h(x_1, x_2) = x_1^2 + x_2^2$ (in the $L_2([-1, 1]^2)$ function space norm $\|g\| \equiv \langle g, g \rangle^{1/2}$). (Note that the functions $1, x_1, x_2$, and x_1x_2 are orthogonal with this inner product.)

A.7 Section 2.2 Exercises

1. (6HW-15) Consider the linear space of functions on $[-\pi, \pi]$ of the form

$$h(t) = a + bt + c \sin t + d \cos t$$

Equip this space with the inner product $\langle u, g \rangle \equiv \int_{-\pi}^{\pi} u(t)g(t)dt$ and norm $\|g\| \equiv \langle g, g \rangle^{1/2}$. Use the Gram-Schmidt process to orthogonalize the set of functions $\{1, t, \sin t, \cos t\}$ and produce an orthonormal basis for the space.

2. (6HW-11) Consider the linear space of functions on $[0, 1]$ of the form

$$h(t) = a + bt + ct^2 + dt^3$$

Equip this space with the inner product $\langle u, g \rangle \equiv \int_0^1 u(t)g(t)dt$ and norm $\|f\| = \langle g, g \rangle^{1/2}$. Use the Gram-Schmidt process to orthogonalize the set of functions $\{1, t, t^2, t^3\}$ and produce an orthonormal basis for the space.

3. (6HW-13) Consider the linear space of functions on $[0, 1]^2$ of the form

$$h(t, s) = a + bt + cs + dt^2 + es^2 + hts$$

Equip this space with the inner product $\langle u, g \rangle \equiv \iint_{[0,1]^2} u(t, s) g(t, s) dt ds$ and

norm $\|g\| = \langle g, g \rangle^{1/2}$. Use the Gram-Schmidt process to orthogonalize the set of functions $\{1, t, s, t^2, s^2, ts\}$ and produce an orthonormal basis for the space.

4. (6E1-11) Consider the space of functions on $[-2, 2]$ corresponding to the kernel $\mathcal{K}(x, z) = 1 + xz \cdot \exp(x + z)$ on $[-2, 2]^2$. (All functions $\mathcal{K}(x, c)$ of x for a $c \in [-2, 2]$ belong to the image of $[-2, 2]$ under the non-linear transform $T(x)(\cdot) = \mathcal{K}(x, \cdot)$.)

a) Show that the functions $g(x) = 1$ and $h(x) = x \exp(x)$ both belong to this image of the transform T .

b) Determine whether or not g and h are orthonormal. If they are not, find an orthonormal basis for the span of $\{g, h\}$.

5. (6E2-13) Consider the function $\mathcal{K}((x, y), (u, v))$ mapping $[-1, 1]^2 \times [-1, 1]^2$ to \mathfrak{R} defined by

$$\mathcal{K}((x, y), (u, v)) = (1 + xu + yv)^2 + \exp\left(-(x - u)^2 - (y - v)^2\right)$$

on its domain.

a) Argue carefully that \mathcal{K} is a legitimate "kernel" function.

b) Pick any two linearly independent elements of the space of functions that are linear combinations of "slices" of the kernel, $\mathcal{K}((x, y), \cdot)$, for an $(x, y) \in [-1, 1]^2$ and find an orthonormal basis for the 2-dimensional linear sub-space they span.

6. (5E1-18) The function

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = \exp(-|x_1 - z_1| - |x_2 - z_2|)$$

mapping $\mathfrak{R}^2 \times \mathfrak{R}^2 \rightarrow \mathfrak{R}$ is a kernel function. Consider three real-valued functions (of $\mathbf{z} \in \mathfrak{R}^2$):

$$T((1, 0))(\mathbf{z}) = \mathcal{K}((1, 0), \mathbf{z}) = \exp(-|1 - z_1| - |z_2|),$$

$$T((0, 1))(\mathbf{z}) = \mathcal{K}((0, 1), \mathbf{z}) = \exp(-|z_1| - |1 - z_2|), \text{ and}$$

$$T((0, 0))(\mathbf{z}) = \mathcal{K}((0, 0), \mathbf{z}) = \exp(-|z_1| - |z_2|)$$

Using the inner product for the linear space of functions mapping $\mathfrak{R}^2 \rightarrow \mathfrak{R}$ defined for kernel slices by $\langle T(\mathbf{x}), T(\mathbf{w}) \rangle_{\mathcal{A}} = \mathcal{K}(\mathbf{x}, \mathbf{w})$, find the projection of $T((0, 0))$ onto the subspace of functions spanned by the two functions $T((1, 0))$ and $T((0, 1))$ (i.e. the set of all linear combinations $c \cdot T((1, 0)) + d \cdot T((0, 1))$ for constants c and d).

7. (5HW-18) Below is a small fake dataset with $p = 2$ and $N = 8$.

x_1	x_2	y
1	0	2.03
0	1	.56
-1	0	-2.21
0	-1	-1.46
2	2	5.78
-1	1	-.72
-2	-2	-6.46
1	-1	1.37

First center the y values and standardize both x_1 and x_2 . (We will abuse notation and use \mathbf{x} and \mathbf{z} to stand for standardized versions of input vectors.) Make use of the kernel function $\mathcal{K}(\mathbf{x}, \mathbf{z}) = \exp(-\|\mathbf{x} - \mathbf{z}\|^2)$ and the mapping $T(\mathbf{x}) = \mathcal{K}(\mathbf{x}, \cdot)$ that associates with input vector $\mathbf{x} \in \mathbb{R}^2$ the function $\mathcal{K}(\mathbf{x}, \cdot) : \mathbb{R}^2 \rightarrow \mathfrak{R}$ (an abstract "feature"). In the (very high-dimensional) space of functions mapping $\mathbb{R}^2 \rightarrow \mathfrak{R}$, the $N = 8$ training set generates an 8-d subspace of functions consisting of all linear combinations of the $T(\mathbf{x}_i)$. Two possible inner products in that subspace are the " L_2 " inner product

$$\langle g, h \rangle_{L_2} = \iint_{\mathbb{R}^2} g(\mathbf{x}) h(\mathbf{x}) d\mathbf{x}$$

and the inner product defined for functions in the range of $T(\cdot)$ by

$$\langle T(\mathbf{x}), T(\mathbf{z}) \rangle_{\mathcal{A}} = \mathcal{K}(\mathbf{x}, \mathbf{z})$$

Apply the first 3 steps of the Gram-Schmidt process to the abstract features of the training data (considered in the order given in the data table) to identify 3 orthonormal functions $\mathbb{R}^2 \rightarrow \mathfrak{R}$ that are linear combinations of $T(\mathbf{x}_1), T(\mathbf{x}_2), T(\mathbf{x}_3)$. Do this first using the L_2 inner product, and then using the kernel-based inner product. Are the two sets of 3 functions the same?

A.8 Section 2.3 Exercises

1. (6HW-15) Consider the 5×4 data matrix

$$\mathbf{X} = \begin{bmatrix} 2 & 4 & 7 & 2 \\ 4 & 3 & 5 & 5 \\ 3 & 4 & 6 & 1 \\ 5 & 2 & 4 & 2 \\ 1 & 3 & 4 & 4 \end{bmatrix}$$

a) Use R and find the QR and singular value decompositions of \mathbf{X} . What are the two corresponding bases for $C(\mathbf{X})$?

- b) Use the singular value decomposition of \mathbf{X} to find the eigen (spectral) decompositions of $\mathbf{X}'\mathbf{X}$ and $\mathbf{X}\mathbf{X}'$ (what are eigenvalues and eigenvectors?).
- c) Find the best $rank = 1$ and $rank = 2$ approximations to \mathbf{X} .

2. (6HW-11) Carry out the steps of Problem 1 above using the matrix

$$\mathbf{X} = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & 2 & 1 \\ 2 & 2 & 1 \end{bmatrix}$$

3. (6E2-15) Here is some simple R code and output for a small $N = 5$ and $p = 4$ dataset.

```
>X
      [,1] [,2] [,3] [,4]
[1,]  0.4  2 -0.5  0
[2,] -0.1  0 -0.3  1
[3,]  0.4  0 -0.1  0
[4,]  0.4  0  0.0 -1
[5,]  0.1  2  0.7  0
>
>svd(X)
$d
[1] 2.8551157 1.4762267 0.9397253 0.3549439
$u
      [,1]      [,2]      [,3]      [,4]
[1,] 0.70256076 0.06562895 0.6458656 -0.2618499
[2,] -0.01458943 0.69768837 0.1798028 0.2661822
[3,] 0.01628552 -0.05282808 0.2689008 0.8815301
[4,] 0.02268773 -0.71093125 0.2403923 0.1625961
[5,] 0.71092586 -0.02664090 -0.6484076 0.2388488
$v
      [,1]      [,2]      [,3]      [,4]
[1,] 0.12929953 -0.23823242 0.403567340 0.8738766
[2,] 0.99014314 0.05282123 -0.005410155 -0.1296041
[3,] 0.05222766 -0.17306746 -0.912659300 0.3665691
[4,] -0.01305627 0.95420275 -0.064475843 0.2918382
```

a) What is the best $rank = 2$ approximation to the 5×4 data matrix (in terms of "Frobenius norm" of the difference between \mathbf{X} and the approximation)?

b) Interpret the fact that by far the largest (in absolute value) number in the first column of the " \mathbf{v} " matrix is .99014314.

A.9 Section 2.4 Exercises

1. (6HW-15) Center the columns of \mathbf{X} from Problem 1 of Section A.8 to make the centered data matrix $\widetilde{\mathbf{X}}$.

a) Find the singular value decomposition of $\widetilde{\mathbf{X}}$. What are the principal component directions and principal components for the data matrix? What are the "loadings" of the first principal component?

b) Find the best $rank = 1$ and $rank = 2$ approximations to $\widetilde{\mathbf{X}}$.

c) Find the eigen decomposition of the sample covariance matrix $\frac{1}{5}\widetilde{\mathbf{X}}'\widetilde{\mathbf{X}}$. Find best 1- and 2-component approximations to this covariance matrix.

d) Now standardize the columns of \mathbf{X} to make the matrix $\widetilde{\widetilde{\mathbf{X}}}$. Repeat parts a), b), and c) using this matrix $\widetilde{\widetilde{\mathbf{X}}}$.

2. (6HW-11) Carry out the steps of Problem 1 above using the matrix \mathbf{X} from Problem 2 of Section A.8.

3. (5HW-14) Consider the small (7×3) fake \mathbf{X} matrix below.

$$\mathbf{X} = \begin{bmatrix} 10 & 10 & .1 \\ 11 & 11 & -.1 \\ 9 & 9 & 0 \\ 11 & 9 & -2.1 \\ 9 & 11 & 2.1 \\ 12 & 8 & -4.0 \\ 8 & 12 & 4.0 \end{bmatrix}$$

(Note, by the way, that $\mathbf{x}_3 \approx \mathbf{x}_2 - \mathbf{x}_1$.)

a) Find the QR and singular value decompositions of \mathbf{X} . Use the latter and give best $rank = 1$ and $rank = 2$ approximations to \mathbf{X} .

b) Subtract column means from the columns of \mathbf{X} to make a centered data matrix. Find the singular value decomposition of this matrix. Is it approximately the same as that in part a)? Give the 3 vectors of the principal component scores. What are the principal components for case 1?

Henceforth consider only the centered data matrix of b).

c) What are the singular values? How do you interpret their relative sizes in this context? What are the first two principal component directions? What are the loadings of the first two principal component directions on x_3 ? What is the third principal component direction? Make scatterplots of 7 points (x_1, x_2) and then 7 points with first coordinate the 1st principal component score and the second the 2nd principal component score. How do these compare? Do you expect them to be similar in light of the sizes of the singular values?

d) Find the matrices $\mathbf{X}\mathbf{v}_j\mathbf{v}_j'$ for $j = 1, 2, 3$ and the best $rank = 1$ and $rank = 2$ approximations to \mathbf{X} . How are the latter related to the former?

e) Compute the (N divisor) 3×3 sample covariance matrix for the 7 cases. Then find its singular value decomposition and its eigenvalue decomposition.

Are the eigenvectors of the sample covariance matrix related to the principal component directions of the (centered) data matrix? If so, how? Are the eigenvalues/singular values of the sample covariance matrix related to the singular values of the (centered) data matrix. If so, how?

4. (5HW-16) Consider the small ($N = 11$) fake $p = 2$ set of predictors in the table below.

x_1	11	12	13	14	13	15	17	16	17	18	19
x_2	18	12	14	16	6	10	14	4	6	8	2

a) Plot raw and standardized versions of 11 predictor pairs (x_1, x_2) on the same set of axes (using different plotting symbols for the two versions and a 1:1 aspect ratio for the plotting). (One can standardize variables in R using the `scale()` function.)

b) Find sample means, sample standard deviations, and the sample correlations for both versions of the predictor pairs.

c) Consider the small (11×2) fake \mathbf{X} matrices corresponding to the raw and standardized versions of the data. Interpret the first principal component direction vectors for the two versions and say why (in geometric terms) they are much different.

5. (5HW-14) The functions

$$\mathcal{K}_1(\mathbf{x}, \mathbf{z}) = \exp\left(-\gamma \|\mathbf{x} - \mathbf{z}\|^2\right) \text{ and}$$

$$\mathcal{K}_2(\mathbf{x}, \mathbf{z}) = (1 + \langle \mathbf{x}, \mathbf{z} \rangle)^d$$

are legitimate kernel functions for choice of $\gamma > 0$ and positive integer d . Find the first two *kernel* principal component vectors for \mathbf{X} in Problem 3 above for each of cases

- \mathcal{K}_1 with two different values of γ (of your choosing), and
- \mathcal{K}_2 for $d = 1, 2$.

If there is anything to interpret (and there may not be) give interpretations of the pairs of principal component vectors for each of the 4 cases. (Be sure to use the vectors for "centered versions" of the function space principal component "direction vectors"/functions.)

6. (6HW-17) The function of $(\mathbf{x}, \mathbf{z}) \in \mathbb{R}^p \times \mathbb{R}^p$ defined by

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = (1 + c \langle \mathbf{x}, \mathbf{z} \rangle)^d$$

for $c > 0$ and positive integer d is well-known to be a kernel function.

a) Argue that indeed \mathcal{K} is a kernel function (is non-negative definite) using the facts from Bishop quoted in Section 1.4.3.

For $d = 2$ consider the $c = 1$ and $c = 2$ cases of this construction for $p = 2$.

b) Describe the sets of functions mapping $\mathfrak{R}^2 \rightarrow \mathfrak{R}$ that comprise the abstract linear spaces associated with the reproducing kernels. What is the dimension of these spaces?

c) Identify for each case a transform $T : \mathfrak{R}^2 \rightarrow \mathfrak{R}^M$ so that

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = \langle T(\mathbf{x}), T(\mathbf{z}) \rangle$$

(an ordinary \mathfrak{R}^M inner product of the transformed data vectors).

d) For \mathbf{x} and \mathbf{z} belonging to \mathfrak{R}^2 find the distances in the two inner product function spaces between $T(\mathbf{x})(\cdot) = \mathcal{K}(\mathbf{x}, \cdot)$ and $T(\mathbf{z})(\cdot) = \mathcal{K}(\mathbf{z}, \cdot)$. (Notice that these are not the same. Metrics implied by the kernels change with the kernels.)

e) Below is a small fake dataset. For the $c = 1$ case, consider these data in the order listed and use as many of the data vectors as necessary to produce a (data-dependent) orthonormal basis for the function space spanned by the $T(\mathbf{x}_i)(\cdot)$. (Use the Gram-Schmidt process in the abstract space.)

x_1	x_2
1	0
0	1
-1	0
0	-1
2	2
-1	1
-2	-2
1	-1

f) Note that the fake dataset of part **e)** is centered in \mathfrak{R}^2 . Find ordinary principal component direction vectors \mathbf{v}_1 and \mathbf{v}_2 and corresponding 8-dimensional vectors of principal component scores for the dataset. Then find the first two kernel principal component vectors corresponding to the $c = 1$ case of \mathcal{K} .

7. (6E1-13) Consider a small fake dataset consisting of $N = 6$ data vectors in \mathfrak{R}^2 and use of a kernel function (mapping $\mathfrak{R}^2 \times \mathfrak{R}^2 \rightarrow \mathfrak{R}$) defined by $\mathcal{K}(\mathbf{x}, \mathbf{z}) =$

$\exp(-3\|\mathbf{x}-\mathbf{z}\|^2)$. The data and Gram matrices are

$$\mathbf{X} = \begin{pmatrix} -1.01 & -.99 \\ -.99 & -1.01 \\ -.01 & .01 \\ 0 & 0 \\ .01 & -.01 \\ 2.00 & 2.00 \end{pmatrix} \approx \begin{pmatrix} -1 & -1 \\ -1 & -1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 2 & 2 \end{pmatrix}$$

$$\text{and } \mathbf{K} = \begin{pmatrix} 1 & .998 & .003 & .003 & .003 & .000 \\ .998 & 1 & .003 & .003 & .003 & .000 \\ .003 & .003 & 1 & .999 & .998 & .000 \\ .003 & .003 & .999 & 1 & .999 & .000 \\ .003 & .003 & .998 & .999 & 1 & .000 \\ .000 & .000 & .000 & .000 & .000 & 1 \end{pmatrix} \approx \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

As it turns out (using the approximate form for \mathbf{K})

$$\mathbf{C} = \mathbf{K} - \frac{1}{6}\mathbf{JK} - \frac{1}{6}\mathbf{KJ} + \frac{1}{36}\mathbf{JKJ}$$

has (approximately) a SVD with two non-zero singular values (namely 2.43 and 1.23) and corresponding vectors of principal components

$$\mathbf{u}_1 = (-.51, -.51, .39, .39, .39, -.15)' \quad \text{and} \quad \mathbf{u}_2 = (-.27, -.27, -.12, -.12, -.12, .9)'$$

Say what both principal components analysis on the raw data and kernel principal components indicate about these data.

8. (6E1-19) Let

$$\mathbf{X} = \begin{bmatrix} 15 & 5 & 1 \\ 15 & 5 & -1 \\ 5 & 15 & 1 \\ 5 & 15 & -1 \\ -5 & -15 & 1 \\ -5 & -15 & -1 \\ -15 & -5 & 1 \\ -15 & -5 & -1 \end{bmatrix} \quad \mathbf{U} = \left(\frac{1}{\sqrt{8}} \right) \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \\ 1 & -1 & 1 \\ 1 & -1 & -1 \\ -1 & 1 & 1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \\ -1 & -1 & -1 \end{bmatrix}$$

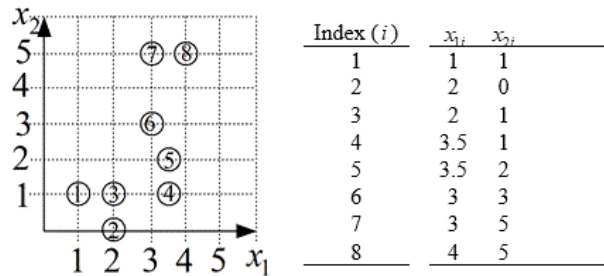
$$\mathbf{D} = \mathit{diag} \begin{pmatrix} 40 \\ 20 \\ 2\sqrt{2} \end{pmatrix} \quad \text{and} \quad \mathbf{V} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

\mathbf{U} , \mathbf{D} , and \mathbf{V} are the elements of the SVD for \mathbf{X} . Use this to answer the following.

- a) Find the best $\text{rank} = 1$ approximation to the matrix \mathbf{X} .
- b) Identify a (3×1) unit vector \mathbf{w} such that the 8 row vectors in \mathbf{X} lie "nearly on" a plane in \mathfrak{R}^3 perpendicular to \mathbf{w} .

c) Give the eigen decomposition of the (8-divisor) sample covariance matrix of a $p = 3$ dataset with cases given by the rows of \mathbf{X} . (Give the 3 eigenvalues and corresponding eigenvectors.)

9. (6E2-15) 6. Below is a small fake $p = 2$ dataset and a scatterplot for it. Consider making graphical spectral features for the dataset, using the symmetric set of index pairs \mathcal{N}_2 (based on 3-nearest-neighbor neighborhoods—a neighborhood including the point itself) and weight function $w(d) = \exp(-d^2)$. Set up an appropriate adjacency matrix and give the 8 node degrees.



10. (5HW-18) Return to the context of Problem 7 of Section A.7. Note that the function $M_{\mathbf{T}} = \frac{1}{8} \sum_{i=1}^8 T(\mathbf{x}_i)$ is a linear combinations of (is in the subspace of functions generated by) the $T(\mathbf{x}_i)$. It makes sense to "center" the abstract features generated by the training set, replacing each $T(\mathbf{x}_i)$ with

$$S(\mathbf{x}_i) = T(\mathbf{x}_i) - M_{\mathbf{T}}$$

a) Compute the matrix

$$\mathbf{C} = (\langle S(\mathbf{x}_i), S(\mathbf{x}_j) \rangle_A)_{\substack{i=1,2,\dots,8 \\ j=1,2,\dots,8}}$$

that is the "centered Gram matrix" for kernel PCA in displays (48) and (49).

c) Do an eigen analysis for the matrix \mathbf{C} . (For Euclidean features, this matrix would be a multiple of a sample covariance matrix.) The eigenvectors of this matrix give kernel principal component scores for the dataset. Consider the first and second of these. To the extent possible, provide interpretations for them.

d) Find the projection of the function $S(.5, .5)$ onto the span of $\{T(\mathbf{x}_i)\}_{i=1,\dots,8}$ in \mathcal{A} and compare contour plots for the function and its projection.

11. (5HW-16) A small example of Prof. Morris involves an $N = 11$ point dataset in the table below.

y	1.003	.807	.669	.628	.554	.511	.531	.502	.610	.701	.942
x	0	.1	.2	.3	.4	.5	.6	.7	.8	.9	1

Center the y values and standardize x . (We will abuse notation and use x and z to stand for standardized versions of input values.)

This question will make use of the kernel function $\mathcal{K}(x, z) = \exp(-.5(x - z)^2)$ and the mapping $T(x)(\cdot) = \mathcal{K}(x, \cdot)$ that associates with input value $x \in \mathfrak{R}$ the function $\mathcal{K}(x, \cdot) : \mathfrak{R} \rightarrow \mathfrak{R}$ (an abstract "feature"). In the (very high-dimensional) space of functions mapping $\mathfrak{R} \rightarrow \mathfrak{R}$, the $N = 11$ training set generates an 11-d subspace of functions consisting of all linear combinations of the $T(x_i)$. As in Problem 10 above set $M_{\mathbf{T}} = \frac{1}{11} \sum_{i=1}^{11} T(x_i)$ and define $S(x_i) = T(x_i) - M_{\mathbf{T}}$.

a) Compute the matrix

$$\mathbf{C} = (\langle S(x_i), S(x_j) \rangle_A) \quad \begin{matrix} i = 1, 2, \dots, 11 \\ j = 1, 2, \dots, 11 \end{matrix}$$

that is the "centered Gram matrix" for kernel PCA in displays (48) and (49).

c) Do an eigen analysis for the matrix \mathbf{C} . (For Euclidean features, this matrix would be a multiple of a sample covariance matrix.) The eigenvectors of this matrix give kernel principal component scores for the dataset. Consider the first and second of these. To the extent possible, provide interpretations for them.

d) Find the projection of the function $S(\cdot)$ onto the span of $\{T(x_i)\}_{i=1, \dots, 11}$ in \mathcal{A} and plot the function and its projection on the same set of axes.

A.10 Section 3.1 Exercises

1. (6HW-11) Consider "data augmentation" methods of penalized least squares fitting.

a) Augment a centered \mathbf{X} matrix with p new rows given by $\sqrt{\lambda} \mathbf{I}_{p \times p}$ and \mathbf{Y} by adding p new entries 0. Argue that OLS fitting with the augmented dataset returns $\hat{\beta}_{\lambda}^{\text{ridge}}$ as a fitted coefficient vector.

b) Show how the elastic net fitted coefficient vector $\hat{\beta}_{\lambda_1, \lambda_2}^{\text{enet}}$ could be found using lasso software and an appropriate augmented dataset.

2. (6E1-11) Consider the $p = 3$ linear prediction problem with $N = 5$ and training data

$$\mathbf{X} = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{20}} \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{20}} \\ 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{20}} \\ -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{20}} \\ 0 & 0 & -\frac{4}{\sqrt{20}} \end{pmatrix} \quad \text{and} \quad \mathbf{Y} = \begin{bmatrix} 2 \\ 3 \\ -1 \\ -1 \\ -3 \end{bmatrix}$$

In answering the following, use the notation that the j th column of \mathbf{X} is \mathbf{x}_j .

- a) Find the fitted OLS coefficient vector $\hat{\beta}^{\text{ols}}$.
 b) For $\lambda = 10$ find the vector $\mathbf{c} \in \mathbb{R}^3$ minimizing

$$\left(\mathbf{Y} - \mathbf{X} \text{diag}(\mathbf{c}) \hat{\beta}^{\text{ols}} \right)' \left(\mathbf{Y} - \mathbf{X} \text{diag}(\mathbf{c}) \hat{\beta}^{\text{ols}} \right) + \lambda \mathbf{1}' \mathbf{c}$$

over choices of \mathbf{c} with non-negative entries.

- c) For $\lambda > 0$ find the fitted ridge coefficient vector, $\hat{\beta}_\lambda^{\text{ridge}}$.
 d) For $\lambda > 0$ find a fitted coefficient vector $\hat{\beta}_\lambda^*$ minimizing $(\mathbf{Y} - \mathbf{X}\mathbf{b})'(\mathbf{Y} - \mathbf{X}\mathbf{b}) + \lambda(b_2^2 + b_3^2)$ as a function of $\mathbf{b} \in \mathbb{R}^3$.
 e) Carefully specify the entire Least Angle Regression path of either $\hat{\mathbf{Y}}$ or $\hat{\beta}$ values.

3. (6E1-13) Consider the $p = 1$ prediction problem with $N = 8$ and training data as below.

$$\mathbf{X} = \begin{pmatrix} 1 & 1 & \sqrt{2} & 0 & 2 & 0 & 0 & 0 \\ 1 & 1 & \sqrt{2} & 0 & -2 & 0 & 0 & 0 \\ 1 & 1 & -\sqrt{2} & 0 & 0 & 2 & 0 & 0 \\ 1 & 1 & -\sqrt{2} & 0 & 0 & -2 & 0 & 0 \\ 1 & -1 & 0 & \sqrt{2} & 0 & 0 & 2 & 0 \\ 1 & -1 & 0 & \sqrt{2} & 0 & 0 & -2 & 0 \\ 1 & -1 & 0 & -\sqrt{2} & 0 & 0 & 0 & 2 \\ 1 & -1 & 0 & -\sqrt{2} & 0 & 0 & 0 & -2 \end{pmatrix} \text{ and } \mathbf{Y} = \begin{pmatrix} 8 \\ 4 \\ 4 \\ 0 \\ 2 \\ 3 \\ 6 \\ 5 \end{pmatrix}$$

Use the notation that the j th column of \mathbf{X} is \mathbf{x}_j .

a) Find the fitted OLS coefficient vector $\hat{\beta}^{\text{ols}}$ for a model including only $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4$ as predictors.

b) Center \mathbf{Y} to create \mathbf{Y}^* and let $\mathbf{x}_j^* = \frac{1}{2\sqrt{2}}\mathbf{x}_j$ for each j . Find $\hat{\beta}^{\text{lasso}} \in \mathbb{R}^7$ optimizing

$$\sum_{i=1}^8 \left(y_i^* - \sum_{j=2}^8 b_j x_{ij}^* \right)^2 + 5 \sum_{i=2}^8 |b_j|$$

over choices of $\mathbf{b} \in \mathbb{R}^7$.

c) The LAR algorithm applied to \mathbf{Y}^* and the set of predictors \mathbf{x}_j^* for $j = 2, 3, \dots, 8$ begins at $\widehat{\mathbf{Y}}^* = \mathbf{0}$ and takes a piecewise linear path through \mathbb{R}^8 to $\widehat{\mathbf{Y}}^{\text{ols}}$. Identify the first two points in \mathbb{R}^8 at which the direction of the path changes, call them \mathbf{W}_1 and \mathbf{W}_2 . (Here you may well wish to use both the connection between the LAR path and the lasso path and explicit formulas for the lasso coefficients.)

4. (5HW-14) Here is a small fake dataset with $p = 4$ and $N = 8$.

y	x_1	x_2	x_3	x_4
3	1	1	1	1
-5	1	1	-1	1
13	1	-1	1	-1
9	1	-1	-1	-1
-3	-1	1	1	-1
-11	-1	1	-1	-1
-1	-1	-1	1	1
-5	-1	-1	-1	1

Notice that the y is centered and the x s are orthogonal (and can easily be made orthonormal by dividing by $\sqrt{8}$). Use the explicit formulas for fitted coefficients in the orthonormal features context to make plots (on a single set of axes for each fitting method, 5 plots in total) of

1. $\hat{\beta}_1, \hat{\beta}_2, \hat{\beta}_3$, and $\hat{\beta}_4$ versus M for best subset (of size M) regression,
2. $\hat{\beta}_1, \hat{\beta}_2, \hat{\beta}_3$, and $\hat{\beta}_4$ versus λ for ridge regression,
3. $\hat{\beta}_1, \hat{\beta}_2, \hat{\beta}_3$, and $\hat{\beta}_4$ versus λ for lasso,
4. $\hat{\beta}_1, \hat{\beta}_2, \hat{\beta}_3$, and $\hat{\beta}_4$ versus λ for $\alpha = .5$ in the elastic net penalty

$$\sum_{i=1}^N (y_i - \hat{y}_i)^2 + \lambda \left((1 - \alpha) \sum_{j=1}^p |\hat{\beta}_j| + \alpha \sum_{j=1}^p \hat{\beta}_j^2 \right)$$

5. $\hat{\beta}_1, \hat{\beta}_2, \hat{\beta}_3$, and $\hat{\beta}_4$ versus λ for the non-negative garrote.

5. (6HW-11) (3.23 of HTF) Suppose that columns of \mathbf{X} with rank p have been standardized, as has \mathbf{Y} . Suppose also that

$$\frac{1}{N} |\langle \mathbf{x}_j, \mathbf{Y} \rangle| = \lambda \quad \forall j = 1, \dots, p$$

Let $\hat{\boldsymbol{\beta}}^{\text{ols}}$ be the usual least squares coefficient vector and $\hat{\mathbf{Y}}^{\text{ols}}$ be the usual projection of \mathbf{Y} onto the column space of \mathbf{X} . Define $\hat{\mathbf{Y}}(\alpha) = \alpha \mathbf{X} \hat{\boldsymbol{\beta}}^{\text{ols}}$ for $\alpha \in [0, 1]$. Find

$$\frac{1}{N} \left| \langle \mathbf{x}_j, \mathbf{Y} - \hat{\mathbf{Y}}(\alpha) \rangle \right| \quad \forall j = 1, \dots, p$$

in terms of α, λ , and $(\mathbf{Y} - \hat{\mathbf{Y}}^{\text{ols}})' (\mathbf{Y} - \hat{\mathbf{Y}}^{\text{ols}})$. Show this is decreasing in α . What is the implication of this as regards the LAR algorithm?

6. (5HW-14) Return to the context of Problem 13 of Section A.2 and the last/largest set of predictors. Center the y vector to produce (say) \mathbf{Y}^* , remove the column of 1s from the \mathbf{X} matrix (giving a 100×9 matrix) and standardize the columns of the resulting matrix, to produce (say) \mathbf{X}^* .

a) Augment \mathbf{Y}^* to \mathbf{Y}^{**} by adding 9 values 0 at the end of the vector (to produce a 109×1 vector) and for value $\lambda = 4$ augment \mathbf{X}^* to \mathbf{X}^{**} (a 109×9 matrix) by adding 9 rows at the bottom of the matrix in the form of $\sqrt{\lambda} \mathbf{I}_{9 \times 9}$. What quantity does OLS based on these augmented data seek to optimize? What is the relationship of this to a ridge regression objective?

b) Use trial and error and matrix calculations based on the explicit form of $\hat{\beta}_\lambda^{\text{ridge}}$ given in Section 3.1.1 to identify a value $\tilde{\lambda}$ for which the error sum of squares for ridge regression is about 1.5 times that of OLS in this problem. Then make a series of at least 5 values from 0 to $\tilde{\lambda}$ to use as candidates for λ . Choose one of these as an "optimal" ridge parameter λ^{opt} here based on 10-fold cross-validation (as was done in Problem 13 of Section A.2). Compute the corresponding predictions \hat{y}_i^{ridge} and plot both them and the OLS predictions as functions of x (connect successive (x, \hat{y}) points with line segments). How do the "optimal" ridge predictions based on the 9 predictors compare to the OLS predictions based on the same 9 predictors?

7. (6E1-13) Consider prediction of a 0/1 (binary) response using a model that says that for two (standardized) predictors z_1 and z_2

$$P[y_i = 1 | (z_{1i}, z_{2i})] = \frac{\exp(\alpha + \beta_1 z_{1i} + \beta_2 z_{2i})}{1 + \exp(\alpha + \beta_1 z_{1i} + \beta_2 z_{2i})}$$

(Training data are N vectors (z_{1i}, z_{2i}, y_i) .) For this problem, one might define a (log-likelihood-based) training error as

$$\overline{\text{err}}(a, b_1, b_2) = \sum_{i=1}^N \ln(1 + \exp(a + b_1 z_{1i} + b_2 z_{2i})) - \sum_{i=1}^N y_i (a + b_1 z_{1i} + b_2 z_{2i})$$

How would you regularize fitting of this model "in ridge regression style" (penalizing only b_1 and b_2 and not a)? Derive 3 equations that you would need to solve simultaneously to carry out regularized fitting.

8. (6E2-13) Suppose that for a pair of positive constants $\lambda_1 \neq \lambda_2$ the predictors \hat{f}_1 and \hat{f}_2 are corresponding ridge regression predictors (their coefficient vectors solve the unconstrained versions of the ridge minimization problem). Is then the predictor

$$\hat{f} = \frac{1}{2} \hat{f}_1 + \frac{1}{2} \hat{f}_2$$

in general a ridge regression predictor? (Make a careful/convincing argument one way or the other.)

9. (6HW-15) Show the equivalence of the two forms of the optimization used to produce the fitted ridge regression parameter. (That is, show that there is a $t(\lambda)$ such that $\hat{\beta}_\lambda^{\text{ridge}} = \hat{\beta}_{t(\lambda)}^{\text{ridge}}$ and a $\lambda(t)$ such that $\hat{\beta}_t^{\text{ridge}} = \hat{\beta}_{\lambda(t)}^{\text{ridge}}$.)

10. (5E1-16) (Ridge regression produces a "grouping effect" for highly correlated predictors) Suppose that in a p -variable SEL prediction problem, input variables x_1, x_2, x_3 have very large absolute correlations. Upon standardization (and arbitrary change of signs of the standardized variables so that all correlations are positive) the variables are essentially the same, and every combination

$$\sum_{j=1}^3 w_j x_j'' \text{ for } w_1, w_2, w_3 \text{ with } w_1 + w_2 + w_3 = 1$$

is essentially the same. So every set of coefficients $\beta_1, \beta_2, \beta_3$ with a given sum $B = \beta_1 + \beta_2 + \beta_3$ has nearly the same $\sum_{j=1}^3 \beta_j x_j''$. Argue then that any minimizer of $\sum_{i=1}^N \left(y_i - \left(\beta_0 + \sum_{j=1}^p \beta_j x_j'' \right) \right)^2 + \lambda \sum_{j=1}^p \beta_j^2$ has $\hat{\beta}_1^{\text{ridge}} \approx \hat{\beta}_2^{\text{ridge}} \approx \hat{\beta}_3^{\text{ridge}}$.

11. (5HW-18) For the situation of Problem 7 of Section A.7 (with centered response and standardized inputs x_1 and x_2) do the following concerning linear predictors

$$\hat{f}(x_1, x_2) = b_1 x_1 + b_2 x_2$$

- a) Plot on the same set of axes the two values b_1 and b_2 as functions of λ (or $\ln \lambda$ if that is easier to compute or interpret) for ridge regression predictors.
- b) Plot on the same set of axes the two values b_1 and b_2 as functions of λ (or $\ln \lambda$ if that is easier to compute or interpret) for lasso regression predictors.

A.11 Section 3.2 Exercises

1. (6E1-11) As it turns out

$$\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{20}} \\ 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{20}} + \frac{1}{\sqrt{20}} \\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} + \frac{1}{\sqrt{20}} \\ -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{20}} \\ 0 & 0 & -\frac{4}{\sqrt{20}} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{20}} \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{20}} \\ 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{20}} \\ -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{20}} \\ 0 & 0 & -\frac{4}{\sqrt{20}} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

Consider a $p = 3$ linear prediction problem where the matrix of training inputs, \mathbf{X} , is the matrix on the left above and $\mathbf{Y}' = (4, 2, 2, 0, 2)$.

- a) Find the single principal component ($M = 1$) fitted coefficient vector $\hat{\beta}^{\text{per}}$.
- b) Find the single component ($M = 1$) partial least squares vector of predictions, $\hat{\mathbf{Y}}^{\text{pls}}$.

2. (6HW-11) Beginning in its Section 5.6, Izenman's book uses an example where PET yarn density is to be predicted from its NIR spectrum. This is a problem where $N = 21$ data vectors \mathbf{x}_j of length $p = 268$ are used to predict the corresponding outputs y_i . Izenman points out that the yarn data are to be found in the `pls` package in R. (The package actually has $N = 28$ cases. Use all of them in the following.) Get those data and make sure that all inputs are standardized and the output is centered. (Use the N divisor for the sample variance.)

a) Using the `pls` package, find the 1, 2, 3, and 4-component PCR and PLS $\hat{\beta}$ vectors.

b) Find the singular values for the matrix \mathbf{X} and use them to plot the function $\text{df}(\lambda)$ for ridge regression. Identify values of λ corresponding to effective degrees of freedom 1, 2, 3, and 4. Find corresponding ridge $\hat{\beta}$ vectors.

c) Plot on the same set of axes $\hat{\beta}_j$ versus j for the PCR, PLS and ridge vectors for number of components/degrees of freedom 1. (Plot them as "functions," connecting consecutive plotted $(j, \hat{\beta}_j)$ points with line segments.) Then do the same for 2, 3, and 4 components/degrees of freedom.

d) It is (barely) possible to find that the best (in terms of R^2) subsets of $M = 1, 2, 3,$ and 4 predictors for OLS are respectively, $\{x_{40}\}, \{x_{212}, x_{246}\}, \{x_{25}, x_{160}, x_{215}\},$ and $\{x_{160}, x_{169}, x_{231}, x_{243}\}$. Find their corresponding coefficient vectors. Use the `lars` package in R and find the lasso coefficient vectors $\hat{\beta}$ with exactly $M = 1, 2, 3,$ and 4 non-zero entries with the largest possible $\sum_{j=1}^{268} |\hat{\beta}_j^{\text{lasso}}|$ (for the counts of non-zero entries).

e) If necessary, re-order/sort the cases by their values of y_i (from smallest to largest) to get a new indexing. Then plot on the same set of axes y_i versus i and \hat{y}_i versus i for ridge, PCR, PLS, best subset, and lasso regressions for number of components/degrees of freedom/number of nonzero coefficients equal to 1. (Plot them as "functions," connecting consecutive plotted (i, y_i) or (i, \hat{y}_i) points with line segments.) Then do the same for 2, 3, and 4 components/degrees of freedom/counts of non-zero coefficients.

f) Use the `glmnet` package in R to do ridge regression and lasso regression here. Find the value of λ for which your lasso coefficient vector in **d)** for $M = 2$ optimizes the quantity

$$\sum_{i=1}^N (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{268} |\hat{\beta}_j|$$

(by matching the error sums of squares). Then, by using the trick of Problem 1 Section A.10 employ the package to find coefficient vectors $\hat{\beta}$ optimizing

$$\sum_{i=1}^N (y_i - \hat{y}_i)^2 + \lambda \left((1 - \alpha) \sum_{j=1}^{268} |\hat{\beta}_j| + \alpha \sum_{j=1}^{268} \hat{\beta}_j^2 \right)$$

for $\alpha = 0, .1, .2, \dots, 1.0$. What effective degrees of freedom are associated with the $\alpha = 1$ version of this? How many of the coefficients β_j are non-zero for each

of the values of α ? Compare error sum of squares for the raw elastic net predictors to that for the linear predictors using (modified elastic net) coefficients

$$(1 + \lambda\alpha) \hat{\beta}_{\lambda,\alpha}^{\text{enet}}$$

3. (5HW-18) For the situation of Problem 11 of Section A.10 find 1-component PCA and PLS predictors.

4. (5E1-14) In a SEL prediction problem with $N = 22$, $p = 5$ standardized predictor variables produce input matrix \mathbf{X} for centered response vector \mathbf{Y} . The singular values of \mathbf{X} are

$$5.970, 5.579, 4.583, 4.132, \text{ and } .397$$

and some matrix products are

$$\mathbf{Y}'\mathbf{X}\mathbf{X}'\mathbf{Y} = 10.27, \mathbf{Y}'\mathbf{X}\mathbf{X}'\mathbf{X}\mathbf{X}'\mathbf{Y} = 312.2, \text{ and} \\ \mathbf{Y}'\mathbf{U} = (-.145, -.53, -.026, .209, -.112)$$

for \mathbf{U} from the singular value decomposition of \mathbf{X} .

a) What are the effective degrees of freedom associated with ridge regression in this context for ridge parameter $\lambda = 2$?

b) Write the $M = 1$ component PCR prediction vector $\hat{\mathbf{Y}}^{\text{PCR}}$ as a function of the first column of \mathbf{U} , say \mathbf{u}_1 .

c) Write the $M = 1$ component PLS prediction vector $\hat{\mathbf{Y}}^{\text{PLS}}$ as a function of the vector $\mathbf{X}\mathbf{X}'\mathbf{Y}$, say \mathbf{w} .

A.12 Section 4.1 Exercises

1. (6E1-17) Below are $N = 8$ training cases (x_i, y_i) for $x \in [0, 1]$ and a corresponding "design matrix" holding values of the first 8 Haar basis functions (in the order $\varphi, \psi, \psi_{1,0}, \psi_{1,1}, \psi_{2,0}, \psi_{2,1}, \psi_{2,2}, \psi_{2,3}$) for the x_i . Consider prediction based on the values of the 8 Haar basis functions.

$$\mathbf{x}_{8 \times 1} = \begin{bmatrix} 1/16 \\ 3/16 \\ 5/16 \\ 7/16 \\ 9/16 \\ 11/16 \\ 13/16 \\ 15/16 \end{bmatrix} \quad \mathbf{y}_{8 \times 1} = \begin{bmatrix} 2 \\ -1 \\ 3 \\ -2 \\ 4 \\ -3 \\ 5 \\ -4 \end{bmatrix} \quad \mathbf{X}_{8 \times 1} = \begin{bmatrix} 1 & 1 & \sqrt{2} & 0 & 2 & 0 & 0 & 0 \\ 1 & 1 & \sqrt{2} & 0 & -2 & 0 & 0 & 0 \\ 1 & 1 & -\sqrt{2} & 0 & 0 & 2 & 0 & 0 \\ 1 & 1 & -\sqrt{2} & 0 & 0 & -2 & 0 & 0 \\ 1 & -1 & 0 & \sqrt{2} & 0 & 0 & 2 & 0 \\ 1 & -1 & 0 & \sqrt{2} & 0 & 0 & -2 & 0 \\ 1 & -1 & 0 & -\sqrt{2} & 0 & 0 & 0 & 2 \\ 1 & -1 & 0 & -\sqrt{2} & 0 & 0 & 0 & -2 \end{bmatrix}$$

a) Find the OLS prediction vector \hat{y}^{ols} here. (This is trivial. Note that the 8 columns of \mathbf{X} are orthogonal.)

- b) Find the 1-component PLS prediction vector \hat{y}^{pls} here.
- c) After normalizing the predictors (so that the \Re^8 norm of each column of the normalized \mathbf{X} is 1) find the lasso prediction vector \hat{y}^{lasso} for the penalty parameter $\lambda = 10$. (Center the vector of responses, remove the first column of the \mathbf{X} and work with an 8×7 vector of inputs.)
- d) Using the normalized version of the predictors referred to in part c) find a vector of coefficients \mathbf{b} that minimizes

$$(\mathbf{y} - \mathbf{X}\mathbf{b})'(\mathbf{y} - \mathbf{X}\mathbf{b}) + \mathbf{b}' \mathit{diag}(0, 0, 0, 4, 4, 4, 4) \mathbf{b}$$

2. (5HW-14) Return to the context of Problem 13 of Section A.2. Make up a matrix of inputs based on x consisting of the values of Haar basis functions up through order $m = 3$. (You will need to take the functions defined on $[0, 1]$ and re-scale their arguments to $[-\pi, \pi]$. For a function $g : [0, 1] \rightarrow \Re$ this is the function $g^* : [-\pi, \pi] \rightarrow \Re$ defined by $g^*(x) = g(\frac{x}{2\pi} + .5)$.) This will produce a 100×16 matrix \mathbf{X}_h .

a) Find $\hat{\beta}^{\text{ols}}$ and plot the corresponding \hat{y} s as a function of x with the data also plotted in scatterplot form.

b) Center y and standardize the columns of \mathbf{X}_h . Find the lasso coefficient vectors $\hat{\beta}$ with exactly $M = 2, 4$, and 8 non-zero entries with the largest possible $\sum_{j=1}^{16} |\hat{\beta}_j^{\text{lasso}}|$ (for the counts of non-zero entries). Plot the corresponding \hat{y} s as a function of x on the same set of axes, with the data also plotted in scatterplot form.

3. (6HW-15) For an $N = 100$ dataset made up for Problem 17 of Section A.2 make up a matrix of inputs based on x consisting of the values of Haar basis functions up through order $m = 3$. This will produce a 100×16 matrix \mathbf{X}_h .

a) Find $\hat{\beta}^{\text{ols}}$ and plot the corresponding \hat{y} s as a function of x with the data also plotted in scatterplot form.

b) Center y and standardize the columns of \mathbf{X}_h . Find the lasso coefficient vectors $\hat{\beta}$ with exactly $M = 2, 4$, and 8 non-zero entries with the largest possible $\sum_{j=1}^{16} |\hat{\beta}_j^{\text{lasso}}|$ (for the counts of non-zero entries). Plot the corresponding \hat{y} s as a function of x on the same set of axes, with the data also plotted in scatterplot form.

4. (5E1-20) Here consider (square integrable) functions on the unit interval $(0, 1)$. Four such functions are

$$g_1(x) = I[0 < x < 1], g_2(x) = I[0 < x < .5], g_3(x) = I[0 < x < .25], g_4(x) = I[.5 < x < .75]$$

(Ignore the values $x = .25, .5$, and $.75$. They have 0 probability and are a nuisance.) Using the " L_2 " inner product defined by $\langle f, g \rangle \equiv \int_0^1 f(x)g(x)dx$ use the Gram-Schmidt process to make four orthonormal functions from these, say $h_1(x), h_2(x), h_3(x), h_4(x)$ and say how they are related to the first 4 Haar basis functions.

A.13 Section 4.2 Exercises

1. **(6HW-11)** Find a set of basis functions for the natural (linear outside the interval (ξ_1, ξ_K)) quadratic regression splines with knots at $\xi_1 < \xi_2 < \dots < \xi_K$.

2. **(6HW-11)** (B-Splines) For $a < \xi_1 < \xi_2 < \dots < \xi_K < b$ consider the B-spline bases of order m , $\{B_{i,m}(x)\}$ defined recursively as follows. For $j < 1$ define $\xi_j = a$, and for $j > K$ let $\xi_j = b$. Define

$$B_{i,1}(x) = I[\xi_i \leq x < \xi_{i+1}]$$

(in case $\xi_i = \xi_{i+1}$ take $B_{i,1}(x) \equiv 0$) and then

$$B_{i,m}(x) = \frac{x - \xi_i}{\xi_{i+m-1} - \xi_i} B_{i,(m-1)}(x) + \frac{\xi_{i+m} - x}{\xi_{i+m} - \xi_{i+1}} B_{i+1,(m-1)}(x)$$

(where we understand that if $B_{i,l}(x) \equiv 0$ its term drops out of the expression above). For $a = -0.1$ and $b = 1.1$ and $\xi_i = (i-1)/10$ for $i = 1, 2, \dots, 11$, plot the non-zero $B_{i,3}(x)$. Consider all linear combinations of these functions. Argue that any such linear combination is piecewise quadratic with first derivatives at every ξ_i . If it is possible to do so, identify one or more linear constraints on the coefficients (call them c_i) that will make $q_c(x) = \sum_i c_i B_{3,i}(x)$ linear to the left of ξ_1 (but otherwise minimally constrain the form of $q_c(x)$).

3. **(5E1-14)** Suppose one desires to fit a function to N data pairs (x_i, y_i) that is linear outside the interval $[0, 1]$, is quadratic in each of the intervals $[0, .5]$ and $[.5, 1]$ and has a first derivative for all x (has no sharp corners). Specify 4 functions $h_1(x), h_2(x), h_3(x)$, and $h_4(x)$ and one linear constraint on coefficients $\beta_0, \beta_1, \beta_2, \beta_3$, and β_4 so that the function

$$y = \beta_0 + \beta_1 h_1(x) + \beta_2 h_2(x) + \beta_3 h_3(x) + \beta_4 h_4(x)$$

is of the desired form.

4. **(6E1-11)** Consider a toy $p = 1$ SEL prediction problem with training data below.

x	-1.0	-.75	-.50	-.25	0	.25	.50	.75	1.0
y	0	2	3	5	4	4	2	2	1

Set up an \mathbf{X} matrix for ordinary multiple linear regression that could be used to fit a linear regression spline with knots at $\xi_1 = -.5, \xi_2 = 0$, and $\xi_3 = .5$. For your set-up, what linear combination of fitted regression parameters produces the prediction at $x = 0$?

5. **(5HW-14)** For the dataset of Problem 13 of Section A.2 make up a 100×7 matrix \mathbf{X}_h of inputs based on x consisting of the values of basis functions for natural cubic splines with knots ξ_j

$$h_1(x) = 1, h_2(x) = x, \text{ and for } j = 1, 2, \dots, K - 2$$

$$h_{j+2}(x) = (x - \xi_j)_+^3 - \left(\frac{\xi_K - \xi_j}{\xi_K - \xi_{K-1}} \right) (x - \xi_{K-1})_+^3 + \left(\frac{\xi_{K-1} - \xi_j}{\xi_K - \xi_{K-1}} \right) (x - \xi_K)_+^3$$

for the $K = 7$ knot values

$$\xi_1 = -3.0, \xi_2 = -2.0, \xi_3 = -1.0, \xi_4 = 0.0, \xi_5 = 1.0, \xi_6 = 2.0, \xi_7 = 3.0$$

Find $\hat{\beta}^{\text{ols}}$ and plot the corresponding natural cubic regression spline, with the data also plotted in scatterplot form.

6. (6HW-15) For the dataset of Problem 17 of Section A.2 make up a 100×7 matrix \mathbf{X}_h of inputs based on x consisting of the values of basis functions for natural cubic splines with knots ξ_j of the general form given in Problem 5 above for the $K = 7$ knot values

$$\xi_1 = 0, \xi_2 = .1, \xi_3 = .3, \xi_4 = .5, \xi_5 = .7, \xi_6 = .9, \xi_7 = 1.0$$

Find $\hat{\beta}^{\text{ols}}$ and plot the corresponding natural cubic regression spline, with the data also plotted in scatterplot form.

7. (6HW-17) Your instructor will provide a dataset giving the maximum numbers of home runs hit by a "big league" professional baseball player in the US for each of 145 consecutive seasons. Consider these as values y_1, y_2, \dots, y_{145} and take $x_i = i$. Consider the basis functions for natural cubic splines with knots ξ_j of the general form in Problem 5 above. Using knots $\xi_j = 2 + (j - 1) / 14$ for $j = 1, 2, \dots, 11$ fit a natural cubic regression spline to the home run data. Plot the fitted function on the same axes as the data points.

A.14 Section 4.3 Exercises

1. (6HW-13) Consider the space of continuous functions on $[0, 1] \times [0, 1]$ that are linear (i.e. are of the form $y = a + bx_1 + cx_2$) on each of the squares

$$S_1 = [0, .5] \times [0, .5], S_2 = [0, .5] \times [.5, 1], S_3 = [.5, 1] \times [0, .5], \text{ and } S_4 = [.5, 1] \times [.5, 1]$$

- Find a set of basis functions for the space described above.
- Your instructor will send you a dataset generated from a model with

$$\mathbf{E}[y|x_1, x_2] = 2x_1x_2$$

Find the best fitting linear combination of the basis functions according to least squares.

c) Describe a set of basis functions for all continuous functions on $[0, 1] \times [0, 1]$ that for

$$0 = \xi_0 < \xi_1 < \xi_2 < \dots < \xi_{K-1} < \xi_K = 1 \quad \text{and} \quad 0 = \eta_0 < \eta_1 < \dots < \eta_{M-1} < \eta_M = 1$$

are linear on each rectangle $S_{km} = [\xi_{k-1}, \xi_k] \times [\eta_{m-1}, \eta_m]$. How many such basis functions are needed to represent these functions?

A.15 Section 5.1 Exercises

1. (6HW-11) Suppose that $a < x_1 < x_2 < \dots < x_N < b$ and $s(x)$ is a natural cubic spline with knots at the x_i interpolating the points (x_i, y_i) (i.e. $s(x_i) = y_i$).

a) Let $z(x)$ be any twice continuously differentiable function on $[a, b]$ also interpolating the points (x_i, y_i) . Show that

$$\int_a^b (s''(x))^2 dx \leq \int_a^b (z''(x))^2 dx$$

(Hint: Consider $d(x) = z(x) - s(x)$, write

$$\int_a^b (d''(x))^2 dx = \int_a^b (z''(x))^2 dx - \int_a^b (s''(x))^2 dx - 2 \int_a^b s''(x) d''(x) dx$$

and use integration by parts and the fact that $s'''(x)$ is piecewise constant.)

b) Use a) and prove that the minimizer of $\sum_{i=1}^N (y_i - h(x_i))^2 + \lambda \int_a^b (h''(x))^2 dx$ over the set of twice continuously differentiable functions on $[a, b]$ is a natural cubic spline with knots at the x_i .

2. (5HW-16) For $p = 1$ suppose that N observations (x_i, y_i) have distinct x_i , and for simplicity of notation, suppose that $x_1 < x_2 < \dots < x_N$. Consider the basis functions for natural cubic splines with K knots ξ_j given in Section 4.2:

$$h_1(x) = 1, h_2(x) = x, \text{ and for } j = 1, 2, \dots, K-2$$

$$h_{j+2}(x) = (x - \xi_j)_+^3 - \left(\frac{\xi_K - \xi_j}{\xi_K - \xi_{K-1}} \right) (x - \xi_{K-1})_+^3 + \left(\frac{\xi_{K-1} - \xi_j}{\xi_K - \xi_{K-1}} \right) (x - \xi_K)_+^3$$

Take $K = N$ and $\xi_j = x_j$ for $j = 1, 2, \dots, N$. Obviously, h_1 and h_2 have second derivative functions that are everywhere 0 and the products of these second derivatives with themselves or 2nd derivatives of other basis functions must have 0 integral from a to b .

Then for $j = 1, 2, 3, \dots, N-2$

$$\begin{aligned} h_{j+2}''(x) &= 6(x - x_j) I[x_j \leq x \leq x_{N-1}] \\ &\quad + 6 \left((x - x_j) - \left(\frac{x_N - x_j}{x_N - x_{N-1}} \right) (x - x_{N-1}) \right) I[x_{N-1} \leq x \leq x_N] \\ &\quad + 6 \left((x - x_j) - \left(\frac{x_N - x_j}{x_N - x_{N-1}} \right) (x - x_{N-1}) + \left(\frac{x_{N-1} - x_j}{x_N - x_{N-1}} \right) (x - x_N) \right) I[x_N \leq x \leq b] \\ &= 6(x - x_j) I[x_j \leq x \leq x_{N-1}] \\ &\quad + 6 \left(x \left(\frac{x_j - x_{N-1}}{x_N - x_{N-1}} \right) + x_{N-1} \left(\frac{x_N - x_j}{x_N - x_{N-1}} \right) - x_j \right) I[x_{N-1} \leq x \leq x_N] \\ &= 6(x - x_j) I[x_j \leq x \leq x_{N-1}] + 6(x - x_N) \left(\frac{x_j - x_{N-1}}{x_N - x_{N-1}} \right) I[x_{N-1} \leq x \leq x_N] \end{aligned}$$

Thus for $j = 1, 2, 3, \dots, N - 2$

$$\begin{aligned} \int_a^b (h''_{j+2}(x))^2 dx &= 12 \left((x_{N-1} - x_j)^3 + (x_N - x_{N-1})^3 \left(\frac{x_{N-1} - x_j}{x_N - x_{N-1}} \right)^2 \right) \\ &= 12 \left((x_{N-1} - x_j)^3 + (x_N - x_{N-1}) (x_{N-1} - x_j)^2 \right) \\ &= 12 (x_{N-1} - x_j)^2 (x_N - x_j) \end{aligned}$$

and for positive integers $1 \leq j < k \leq N - 2$

$$\begin{aligned} \int_a^b h''_{j+2}(x) h''_{k+2}(x) dx &= 36 \left(\int_{x_k}^{x_{N-1}} (x - x_j)(x - x_k) dx + \int_{x_{N-1}}^{x_N} (x - x_N)^2 \frac{(x_j - x_{N-1})(x_k - x_{N-1})}{(x_N - x_{N-1})^2} dx \right) \\ &= 36 \left(\frac{(x_{N-1} - x_k)^3}{3} + (x_k - x_j) \frac{(x_{N-1} - x_k)^2}{2} \right) - 36 \left(\frac{(x_j - x_{N-1})(x_k - x_{N-1})}{(x_N - x_{N-1})^2} \right) \frac{(x_{N-1} - x_N)^3}{3} \\ &= 6 (x_{N-1} - x_k)^2 (2(x_{N-1} - x_k) + 3(x_k - x_j)) - 12 (x_j - x_{N-1})(x_k - x_{N-1})(x_{N-1} - x_N) \\ &= 6 (x_{N-1} - x_k)^2 (2x_{N-1} + x_k - 3x_j) + 12 (x_{N-1} - x_k)(x_{N-1} - x_j)(x_N - x_{N-1}) \end{aligned}$$

Do the smoothing spline computations for the dataset of Problem 11 Section A.9 "from scratch" using the above representations of the entries of the matrix $\mathbf{\Omega}$. That is,

a) Compute the 11×11 matrix $\mathbf{\Omega}$.

b) For $\lambda = 1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$, and 0 compute the smoother matrices \mathbf{S}_λ and the effective degrees of freedom.

c) Find the penalty matrix \mathbf{K} and its eigen decomposition. Plot as functions of x_i (or just i assuming that you have ordered the values of x) the entries of the eigenvectors of this matrix (connect successive points with line segments so that you can see how these change in character as the corresponding eigenvalue of \mathbf{K} increases—the corresponding eigenvalue of \mathbf{S}_λ decreases). Which \mathfrak{R}^{11} components of the observed \mathbf{Y} are most suppressed in the smoothing operation? Can you describe them in qualitative terms?

A.16 Section 5.2 Exercises

1. (6HW-11) A $p = 2$ dataset provided with these notes consists of $N = 441$ training vectors (x_{1i}, x_{2i}, y_i) for the distinct pairs (x_{1i}, x_{2i}) in the set $\{-1.0, -0.9, \dots, 0.9, 1.0\}^2$ where the y_i were generated as

$$y_i = \frac{\sin(10(x_{1i} + x_{2i}))}{10(x_{1i} + x_{2i})} + \epsilon_i$$

(with the convention that $\sin(0)/0 = 1$) for iid $N(0, (.02)^2)$ variables ϵ_i .

a) Why should you expect MARS to be ineffective in producing a predictor in this context? (You may want to experiment with the `earth` package in R trying out MARS.)

b) Fit a thin plate spline to these data using the `Tps` function in the `fields` package. Contour plot your results.

2. (6HW-13) A $p = 2$ dataset provided with these notes consists of $N = 81$ training vectors (x_{1i}, x_{2i}, y_i) for pairs (x_{1i}, x_{2i}) in the set $\{-2.0, -1.5, \dots, 1.5, 2.0\}^2$ where the y_i were generated as

$$y_i = (x_{1i}^2 + x_{2i}^2) / (1 + (x_{1i}^2 + x_{2i}^2)) + \epsilon_i$$

for iid $N(0, (.1)^2)$ variables ϵ_i . Use it in the following.

a) Why should you expect MARS to be ineffective in producing a predictor in this context? (You may want to experiment with the `earth` package in R trying out MARS.)

b) Fit a thin plate spline to these data using the `Tps` function in the `fields` package.

A.17 Section 5.3 Exercises

1. (6E1-13) Return to the scenario of Problem 3 of Section A.10.

a) Find $\hat{\mathbf{Y}}^{\text{penalty}} \in \mathfrak{R}^8$ optimizing

$$(\mathbf{Y} - \mathbf{v})'(\mathbf{Y} - \mathbf{v}) + \langle \mathbf{v}, \mathbf{x}_2^* \rangle^2 + 2 \left(\langle \mathbf{v}, \mathbf{x}_3^* \rangle^2 + \langle \mathbf{v}, \mathbf{x}_4^* \rangle^2 \right) + 4 \sum_{j=5}^8 \langle \mathbf{v}, \mathbf{x}_j^* \rangle^2$$

over choices of $\mathbf{v} \in \mathfrak{R}^8$.

b) Find an 8×8 smoother matrix \mathbf{S} corresponding to the penalty in **a)** (a matrix so that for any $\mathbf{Y} \in \mathfrak{R}^8$ a $\hat{\mathbf{Y}}^{\text{penalty}}$ optimizing the form in part **a)** is $\mathbf{S}\mathbf{Y}$) and plot values in the 4th row of this matrix against $x - .500$.

A.18 Section 6.1 Exercises

1. (6HW-11) Suppose that with $p = 1$,

$$y|x \sim N\left(\frac{\sin(12(x + .2))}{x + .2}, 1\right)$$

and $N = 101$ training data pairs are available with $x_i = (i - 1) / 100$ for $i = 1, 2, \dots, 101$. A dataset like this is provided with these notes. Use it in the following.

a) Fit all of the following using first 5 and then 9 effective degrees of freedom

- a cubic smoothing spline,
- a locally weighted linear regression smoother based on a normal density kernel, and
- a locally weighted linear regression smoother based on a tri-cube kernel.

Plot for 5 effective degrees of freedom all of y_i and the 3 sets of smoothed values against x_i . Connect the consecutive (x_i, \hat{y}_i) for each fit with line segments so that they plot as "functions." Then redo the plotting for 9 effective degrees of freedom.

b) For all of the fits in **a)** plot as a function of i the coefficients c_i applied to the observed y_i in order to produce $\hat{f}(x) = \sum_{i=1}^{101} c_i y_i$ for $x = .05, .1, .2, .3$. (Make a different plot of three curves for 5 degrees of freedom and each of the values x (four in all). Then redo the plotting for 9 degrees of freedom.)

2. (6HW-13) Suppose that with $p = 1$,

$$y|x \sim N\left(\sin\left(\frac{1.5}{x+.1}\right) + \exp(-2x), (.5)^2\right)$$

(the conditional standard deviation is $.5$) and $N = 101$ training data pairs are available with $x_i = (i - 1)/100$ for $i = 1, 2, \dots, 101$. A dataset like this is provided with these notes. Use it in place of the dataset described in Problem 1 above and redo all of that problem.

3. (6E1-11) Suppose that P is such that x has pdf

$$p(x) = \frac{3}{2}I\left[0 < x < \frac{1}{2}\right] + \frac{1}{2}I\left[\frac{1}{2} < x < 1\right] \quad \text{on } [0, 1]$$

and the conditional distribution of $y|x$ is $N(x, 1)$. Suppose training data (x_i, y_i) for $i = 1, \dots, N$ are iid P and that with ϕ the standard normal pdf, one uses the Nadaraya-Watson estimator for $E[y|x = .5] = .5$,

$$\hat{f}(.5) = \frac{\sum_{i=1}^N y_i \phi(.5 - x_i)}{\sum_{i=1}^N \phi(.5 - x_i)}$$

Use the law of large numbers and the continuity of the ratio function and write out the (in probability) limit for $\hat{f}(.5)$ in terms of a ratio of two definite integrals and then argue that the limit is not $.5$.

4. (6E1-11) Consider a toy problem where one is to employ locally weighted straight line regression smoothing based on the Epanechnikov quadratic kernel in a $p = 1$ context with training data given in Problem 4 of Section A.11. Using a bandwidth of $\lambda = .5$, give a small (augmented) dataset for which ordinary simple linear regression (OLS) will produce the smoothed prediction at $x = 0$ (that is, $\hat{f}_{.5}(0)$) for the original training data.

5. (6E1-13) Return to the scenario of Problem 1 of Section A.17. If one accepts the statistical conventional wisdom that (generalized) "spline" smoothing is

nearly equivalent to kernel smoothing, in light of your plot in **b)** of that problem identify a kernel that might provide smoothed values similar to those for the penalty used there. (Name a kernel and choose a bandwidth.)

6. (6HW-15) In a $p = 1$ smoothing context like that of Problem 2 of Section A.15, where $N = 11$ training data pairs (x_i, y_i) have $x_1 = 0, x_2 = .1, x_3 = .2, \dots, x_{11} = 1.0$, consider locally weighted linear regression based on a Gaussian kernel.

a) Compute and plot effective degrees of freedom as a function of the bandwidth, λ . (It may be most effective to make the plot with λ on a log scale or some such.) Do simple numerical searches to identify values of λ corresponding to effective degrees of freedom 2.5, 3, 4, and 5.

b) Compare the smoothing matrix " \mathbf{S}_λ " for Problem 2 of Section A.15 for 4 effective degrees of freedom to the matrix " \mathbf{L}_λ " in the present context also producing 4 effective degrees of freedom. What is the 11×11 matrix difference? Plot, as a function of column index, the values in the 1st, 3rd, and 5th rows of the two matrices, connecting with line segments successive values from a given row. (Connect consecutive plotted points for a given row of a given matrix and use different plotting symbols, colors, and/or line weights and types so that you can make qualitative comparisons of the nature of these.)

7. (6E1-17) Consider a 1-d N-W smoothing problem on $[0, 2]$ for values of $x_j = .1(j - 1)$ for $j = 1, 2, \dots, 21$. Suppose that one uses weights

$$w(|i - j|) = \begin{cases} .5 & \text{if } i = j \\ .25 & \text{if } |i - j| = 1 \\ 0 & \text{otherwise} \end{cases}$$

to make smoothed values

$$\hat{y}_j = \sum_{i=1}^{21} w(|i - j|) y_i$$

except for the "edge" cases where we'll take $\hat{y}_1 = .5y_1 + .5y_2$ and $\hat{y}_{21} = .5y_{20} + .5y_{21}$.

a) For \mathbf{S} the smoother matrix to be applied to a vector of observations $\mathbf{Y} = (y_1, y_2, \dots, y_{21})$ to get smoothed values, what are effective degrees of freedom?

b) What are (except for the "edge" cases, now with indices $j = 1, 2, 20$, and 21) the weights, say $w_2(|i - j|)$, used to make "doubly smoothed" values via two successive applications of the original smoothing. That is, for $\hat{\mathbf{Y}} = \mathbf{S}\mathbf{S}\mathbf{Y}$? What (approximately, you don't need to get exactly the right terms for the edge cases) are effective degrees of freedom for $\mathbf{S}\mathbf{S}$?

c) Consider local linear regression in this same context, where the original weights are used and thus (except for edge cases) the slope and intercept used to make \hat{y}_j are determined by minimizing

$$.25(y_{j-1} - (\beta_0 + \beta_1 x_{j-1}))^2 + .5(y_j - (\beta_0 + \beta_1 x_j))^2 + .25(y_{j+1} - (\beta_0 + \beta_1 x_{j+1}))^2$$

(or equivalently 4 times this quantity). Ultimately (again except for edge cases) what weights go into a smoother matrix for an "equivalent N-W kernel smoother" in this case? (It may be helpful to recall that OLS for SLR produces $b_1 = \left(\sum_{i=1}^N (y_i - \bar{y})(x_i - \bar{x}) \right) / \left(\sum_{i=1}^N (x_i - \bar{x})^2 \right)$ and $b_0 = \bar{y} - b_1\bar{x}$.)

d) Use R to compute the n th power of \mathbf{S} for a reasonably large n . Why is this form really no surprise?

8. (6E1-15) Here is (a rounded version of) a smoother matrix \mathbf{S}_λ , for a N-W smoother with Gaussian kernel for data with $\mathbf{x}' = (0, 0.1, 0.2, \dots, 0.8, 0.9, 1.0)$.

```

      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
[1,]  0.47 0.35 0.14 0.03 0.00 0.00 0.00 0.00 0.00 0.00 0.00
[2,]  0.26 0.35 0.26 0.11 0.02 0.00 0.00 0.00 0.00 0.00 0.00
[3,]  0.10 0.23 0.31 0.23 0.10 0.02 0.00 0.00 0.00 0.00 0.00
[4,]  0.02 0.09 0.23 0.31 0.23 0.09 0.02 0.00 0.00 0.00 0.00
[5,]  0.00 0.02 0.09 0.23 0.31 0.23 0.09 0.02 0.00 0.00 0.00
[6,]  0.00 0.00 0.02 0.09 0.23 0.31 0.23 0.09 0.02 0.00 0.00
[7,]  0.00 0.00 0.00 0.02 0.09 0.23 0.31 0.23 0.09 0.02 0.00
[8,]  0.00 0.00 0.00 0.00 0.02 0.09 0.23 0.31 0.23 0.09 0.02
[9,]  0.00 0.00 0.00 0.00 0.00 0.02 0.10 0.23 0.31 0.23 0.10
[10,] 0.00 0.00 0.00 0.00 0.00 0.00 0.02 0.11 0.26 0.35 0.26
[11,] 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.03 0.14 0.35 0.47

```

a) Approximately what bandwidth (λ) and effective degrees of freedom are associated with this matrix?

b) A rounded version of the matrix product $\mathbf{S}_\lambda \mathbf{S}_\lambda$ is below. Thinking of this product as itself a smoother matrix, what might you think of as "an equivalent kernel"? (Give values of weights $w(|i - j|)$ for i, j indices 1 to 11 so that $\hat{y}_j \approx \sum_{i=1}^{11} w(|i - j|) y_i$.)

```

      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
[1,]  0.33 0.32 0.21 0.10 0.03 0.01 0.00 0.00 0.00 0.00 0.00
[2,]  0.24 0.28 0.24 0.14 0.07 0.02 0.01 0.00 0.00 0.00 0.00
[3,]  0.14 0.21 0.24 0.20 0.12 0.06 0.02 0.01 0.00 0.00 0.00
[4,]  0.06 0.13 0.19 0.22 0.19 0.12 0.06 0.02 0.01 0.00 0.00
[5,]  0.02 0.06 0.12 0.19 0.22 0.19 0.12 0.06 0.02 0.01 0.00
[6,]  0.01 0.02 0.06 0.12 0.19 0.22 0.19 0.12 0.06 0.02 0.01
[7,]  0.00 0.01 0.02 0.06 0.12 0.19 0.22 0.19 0.12 0.06 0.02
[8,]  0.00 0.00 0.01 0.02 0.06 0.12 0.19 0.22 0.19 0.13 0.06
[9,]  0.00 0.00 0.00 0.01 0.02 0.06 0.12 0.20 0.24 0.21 0.14
[10,] 0.00 0.00 0.00 0.00 0.01 0.02 0.07 0.14 0.24 0.28 0.24
[11,] 0.00 0.00 0.00 0.00 0.00 0.01 0.03 0.10 0.21 0.32 0.33

```

c) Here is a bit of R code and more output for this problem.

```

>round(eigen(S)$values,3)
[1] 1.000 0.921 0.730 0.509 0.317 0.176 0.087 0.038 0.015 0.005 0.001
>round(eigen(S)$vectors[,1],3)

```


[1] -0.302 -0.302 -0.302 -0.302 -0.302 -0.302 -0.302 -0.302 -0.302 -0.302
 -0.302

While \mathbf{S}_λ is not symmetric, it is non-singular and has 11 real eigenvalues $1 = d_1 > d_2 > \dots > d_{11} > 0$ with corresponding linearly independent unit eigenvectors $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_{11}$ such that $\mathbf{S}_\lambda \mathbf{u}_j = d_j \mathbf{u}_j$. So with $\mathbf{U} = (\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_{11})$ and $\mathbf{D} = \mathbf{diag}(d_1, d_2, \dots, d_{11})$ we have $\mathbf{S}_\lambda \mathbf{U} = \mathbf{UD}$ and $\mathbf{S}_\lambda = \mathbf{UDU}^{-1}$. The output above provides the eigenvalues and \mathbf{u}_1 .

The n th power of \mathbf{S}_λ , \mathbf{S}_λ^n , has a limit. What is it? Argue that your answer is correct. What are the corresponding limits of $\mathbf{S}_\lambda^n \mathbf{Y}$ and of the effective degrees of freedom of \mathbf{S}_λ^n ?

9. (6HW-17) Consider again the home run dataset of Problem 7 Section A.13. Fit with first approximately 5 and then 9 effective degrees of freedom

- a cubic smoothing spline (using `smooth.spline()`), and
- a locally weighted linear regression smoother based on a tri-cube kernel (using `loess(...,span= ,degree=1)`) to the home run data.

Plot for approximately 5 effective degrees of freedom all of y_i and the 2 sets of smoothed values against x_i . Connect the consecutive (x_i, \hat{y}_i) for each fit with line segments so that they plot as "functions." Then redo the plotting for 9 effective degrees of freedom.

10. (6HW-19) Consider again the fake data of Problem 2 of Section A.15. Carry out the steps of Problem 9 above on this dataset.

11. (5E1-14) Below is a particular smoother matrix, \mathbf{S} , for $p = 1$ data at values $x = 0, .1, .2, .3, \dots, .9, 1.0$ (The labeling convention used below is $x_1 = 0, x_2 = .1, x_3 = .2, \dots, x_{11} = 1.0$.)

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]	[,10]	[,11]
[1,]	.721	.265	.013	.000	.000	.000	.000	.000	.000	.000	.000
[2,]	.210	.570	.210	.010	.000	.000	.000	.000	.000	.000	.000
[3,]	.010	.208	.564	.208	.010	.000	.000	.000	.000	.000	.000
[4,]	.000	.010	.208	.564	.208	.010	.000	.000	.000	.000	.000
[5,]	.000	.000	.010	.208	.564	.208	.010	.000	.000	.000	.000
[6,]	.000	.000	.000	.010	.208	.564	.208	.010	.000	.000	.000
[7,]	.000	.000	.000	.000	.010	.208	.564	.208	.010	.000	.000
[8,]	.000	.000	.000	.000	.000	.010	.208	.564	.208	.010	.000
[9,]	.000	.000	.000	.000	.000	.000	.010	.208	.564	.208	.010
[10,]	.000	.000	.000	.000	.000	.000	.000	.010	.210	.570	.210
[11,]	.000	.000	.000	.000	.000	.000	.000	.000	.013	.265	.721

- What effective degrees of freedom are associated with this smoother?
- Approximately what bandwidth is associated with this smoother?

c) For training data as below, what is $\hat{f}(.4)$?

y	1	3	2	4	2	6	7	9	7	8	6
x	0	.1	.2	.3	.4	.5	.6	.7	.8	.9	1

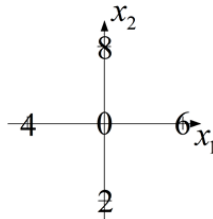
A.19 Section 6.2 Exercises

1. **(6HW-13)** Apply 2-d locally weighted regression smoothing on the dataset of Problem 1 of Section A.16 using the `loess()` function in R. "Surface plot/perspective plot" this for 2 different choices of smoothing parameters along with both the raw data and the mean function. (If nothing else, JMP will do this under its "Graph" menu.)

2. **(6HW-13)** Apply 2-d locally weighted regression smoothing on the dataset of Problem 2 of Section A.16 using the `loess()` function in R. "Surface plot/perspective plot" this for 2 different choices of smoothing parameters along with both the raw data and the mean function.

3. **(5E1-16)** Consider the small ($N = 5$) training set for a $p = 2$ SEL prediction problem given in the table below and represented in the corresponding plot.

x_1	x_2	y
-1	0	4
0	-1	2
0	0	0
0	1	8
1	0	6



a) Find the OLS predictor of y of the form $\hat{y} = \hat{f}(\mathbf{x}) = b_0 + b_1x_1 + b_2x_2$. Show "by hand" calculations. Note that predictors x_1 and x_2 can be standardized to $x'_1 = \sqrt{\frac{5}{2}}x_1$ and $x'_2 = \sqrt{\frac{5}{2}}x_2$ and made orthonormal as $x''_1 = \sqrt{\frac{1}{2}}x_1$ and $x''_2 = \sqrt{\frac{1}{2}}x_2$.

b) Consider the penalized least squares problem of minimizing (for orthonormal predictors x''_1 and x''_2) the quantity

$$\sum_{i=1}^5 (y_i - (\beta_0 + \beta_1x''_1 + \beta_2x''_2))^2 + \lambda (|\beta_1| + |\beta_2|)$$

Plot on the same set of axes minimizers $\hat{\beta}_1^{\text{lasso}}$ and $\hat{\beta}_2^{\text{lasso}}$ as functions of λ .

c) Evaluate the first PLS component \mathbf{z}_1 in this problem and find $\hat{\beta}^{\text{pls}} \in \mathbb{R}^2$ (for centered y values and standardized predictors so that the matrix of predictors \mathbf{x}' is 5×2) so that $\hat{Y}^{\text{pls}} = \mathbf{X}\hat{\beta}^{\text{pls}}$ for a 1-component PLS predictor. Show "by hand" calculations.

d) Since standardization requires multiplying x_1 and x_2 by the same constant, the 3-nn predictor here is the same whether computed on the raw (x_1, x_2) values or after standardization. What is it? (It takes on only a few different values. Give those values and specify the regions in which they pertain in terms of the original variables.)

e) (Again, since standardization requires multiplying x_1 and x_2 by the same constant) 2-d kernel smoothing methods applied on original and standardized scales are equivalent. So consider locally weighted bivariate regression done on the original scale using the Epanechnikov quadratic kernel and bandwidth $\lambda = 1$. Write out (in completely explicit terms) the sum to be optimized by choice of constants $\beta_0, \beta_1, \beta_2$ in order to produce a prediction of the form $\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$ for the input vector $(\frac{1}{2}, \frac{1}{2})$. What is the value of this prediction?

A.20 Section 7.1 Exercises

1. (6HW-11) Consider again the situation of Problem 1 Section A.16. If you were going to use a structured kernel and 1-d smoothing to produce a predictor here, what form for the matrix \mathbf{A} would work best? What would be a completely ineffective choice of a matrix \mathbf{A} ? Use the good choice of \mathbf{A} and produce a corresponding set of predictions.

A.21 Section 8.1 Exercises

1. (6HW-11) Use JMP to do neural net fitting (with logistic sigmoidal function, $\sigma(\cdot)$) for the dataset in Problem 1 of Section A.18.

a) Find a neural net with an error sum of squares about like those for the 9 degrees of freedom fits in Section A.18. Provide appropriate JMP reports/summaries. You'll be allowed to vary the number of hidden nodes for a single-hidden-layer architecture and to vary a weight for a penalty made from a sum of squares of coefficients. Each run of the routine makes several random starts of an optimization algorithm. Extract the coefficients from the JMP run and use them to plot the fitted function of x that you settle on. How does this compare to the plotted fits produced in Problem 1 of Section A.18?

b) Try to reproduce what you got from JMP in a) using the R package `neuralnet` (or any other you find that to work better).

2. (6HW-13) Carry out the steps of Problem 1 above on the data of Problem 2 of Section A.18.

3. (5HW-14) Return to the dataset of Problem 2 of Section A.16. Use the neural network routines in JMP to fit the data to get an error sum of squares like you got in Problem 2 of Section A.16. How complicated does the network architecture have to be in order to do a good job fitting these data? Contour or surface plot your fits.

4. (5HW-14) Use all of MARS, thin plate splines, local kernel-weighted linear regression, and neural nets to fit predictors to both the noiseless and the

noisy "hat data." For those methods for which it's easy to make contour or surface plots, do so. Which methods seem most effective on this particular dataset/function?

5. (6E1-11) Consider a $p = 2$ prediction problem with continuous univariate output y . Two possible methods of prediction are under consideration, namely

1. a neural net with single hidden layer and $M = 2$ hidden nodes (and single output node) using $\sigma(u) = 1/(1 + \exp(u))$ and $g(v) = v$, and
2. a projection pursuit regression predictor with $M = 2$ summands $g_m(\mathbf{w}'_m \mathbf{x})$ (based on cubic smoothing splines).

a) Argue carefully that in general, possibility 2 provides more flexibility in fitting than possibility 1.

b) Note that unit vectors in \mathfrak{R}^2 can be parameterized by a single real variable $\theta \in (-\pi, \pi]$. How would you go about choosing a version of possibility 2 that might be expected to provide only "about as much flexibility in fitting" as possibility 1? (This will have to amount to some speculation, but make a sensible suggestion based on "parameter counts.")

6. (6E1-13) Consider approximations to "simple functions" (linear combinations of step functions) using single layer feed-forward neural network forms. First say how you might produce an approximation of a function on \mathfrak{R}^1 that is an indicator function of any interval, $I = (a, b)$ (finite or infinite), say $I[a < x < b]$. Then argue that it's possible to approximate any function of the form $g(x) = \sum_{l=1}^M c_l I[a_l < x < b_l]$ on \mathfrak{R}^1 using a neural network form.

7. (6HW-15) Again use the dataset of Problem 17 of Section A.2.

- a)** Fit with approximately 5 and then 9 effective degrees of freedom
 - i)** a cubic smoothing spline (using `smooth.spline()`), and
 - ii)** a locally weighted linear regression smoother based on a tri-cube kernel (using `loess(..., span=, degree=1)`).

Plot for approximately 5 effective degrees of freedom all of y_i and the 2 sets of smoothed values against x_i . Connect the consecutive (x_i, \hat{y}_i) for each fit with line segments so that they plot as "functions." Then redo the plotting for 9 effective degrees of freedom.

b) Produce a single hidden layer neural net fit with an error sum of squares about like those for the 9 degrees of freedom fits using `nnet()`. You may need to vary the number of hidden nodes for a single-hidden-layer architecture and vary the weight for a penalty made from a sum of squares of coefficients in order to achieve this. For the function that you ultimately fit, extract the coefficients and plot the fitted mean function. How does it compare to the plots made in **a)**?

c) Each run of `nnet()` begins from a different random start and can produce a different fitted function. Make 5 runs using the architecture and penalty

parameter (the "decay" parameter) you settle on for part **b**) and save the 100 predicted values for the 10 runs into 10 vectors. Make a scatterplot matrix of pairs of these sets of predicted values. How big are the correlations between the different runs?

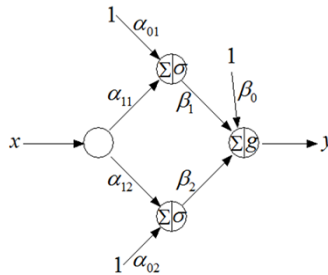
d) Use the `avNNet()` function from the `caret` package to average 20 neural nets with your parameters from part **b**).

8. (6E2-15) Consider a $p = 3$ predictor 2-class neural net classifier, with a single hidden layer having only 2 nodes.

a) Provide the network diagram for this situation and a corresponding likelihood term that might be associated with a training vector $(x_{1i}, x_{2i}, x_{3i}, y_i)$ where y has the -1 versus 1 coding.

b) Suppose that the inputs have been standardized, and completely specify a lasso-motivated jointly continuous prior distribution for the model parameters that might be expected to promote posterior sparsity/near-sparsity for the model parameters.

9. (5E1-16) Below is a toy diagram for a very simple single hidden layer "neural network" mean function of $x \in \mathfrak{R}$ (i.e. $p = 1$). Suppose that outputs/responses y are essentially 3 if $x < 17$ and essentially 8 if $17 < x < 20$, and essentially 3 if $x > 20$. Identify numerical values of neural network parameters $\alpha_{01}, \alpha_{11}, \alpha_{02}, \alpha_{12}, \beta_0, \beta_1, \beta_2$ for which the corresponding predictor is a good approximation of the output mean function. (Here, $\sigma(u) = 1/(1 + \exp(-u))$ and $g(z) = z$.)



10. (5HW-16) Carry out the steps of Problem 7 of this section using the dataset of Problem 13 of Section A.2.

11. (5E1-14) A two-hidden-layer (with 2 nodes per hidden layer) single-input-single-output feed-forward neural network with "activation function" $\sigma(u) = \tanh(u)$ for a $p = 1$ prediction problem is fit to a particular $N = 100$ training set. In notation like that used on Figure 25 this fitting results in

$$\begin{aligned} \hat{\alpha}_{01}^2 &= .005, \hat{\alpha}_{x1}^2 = -.082, \hat{\alpha}_{02}^2 = .023, \hat{\alpha}_{x2}^2 = -.036 \\ \hat{\alpha}_{01}^1 &= -.0007, \hat{\alpha}_{11}^1 = .0004, \hat{\alpha}_{21}^1 = -.0023, \hat{\alpha}_{02}^1 = -.0037, \hat{\alpha}_{12}^1 = -.0155, \hat{\alpha}_{22}^1 = .0356 \\ \hat{\beta}_0 &= 1413, \hat{\beta}_1 = -50513, \hat{\beta}_2 = 850321 \end{aligned}$$

Plot the SEL predictor of y implied by this set of fitted coefficients, $\hat{f}(x)$.

A.22 Section 8.2 Exercises

1. (6HW-11) Consider radial basis functions built from kernels. In particular, consider the choice $D(t) = \phi(t)$, the standard normal pdf.

a) For $p = 1$, plot on the same set of axes the 11 functions

$$\mathcal{K}_\lambda(x, \xi_j) = D\left(\frac{|x - \xi_j|}{\lambda}\right) \text{ for } \xi_j = \frac{j-1}{10} \quad j = 1, 2, \dots, 11$$

first for $\lambda = .1$ and then (in a separate plot) for $\lambda = .01$. Then make plots on the a single set of axes the 11 normalized functions

$$\mathcal{N}_{\lambda_j}(x) = \frac{\mathcal{K}_\lambda(x, \xi_j)}{\sum_{l=1}^{11} \mathcal{K}_\lambda(x, \xi_l)}$$

first for $\lambda = .1$, then in a separate plot for $\lambda = .01$.

b) For $p = 2$, consider the 121 basis functions

$$\mathcal{K}_\lambda(\mathbf{x}, \xi_{ij}) = D\left(\frac{\|\mathbf{x} - \xi_{ij}\|}{\lambda}\right) \text{ for } \xi_{ij} = \left(\frac{i-1}{10}, \frac{j-1}{10}\right) \quad i = 1, \dots, 11 \text{ and } j = 1, \dots, 11$$

Make contour plots for $\mathcal{K}_{.1}(\mathbf{x}, \xi_{6,6})$ and $\mathcal{K}_{.01}(\mathbf{x}, \xi_{6,6})$. Then define

$$\mathcal{N}_{\lambda_{ij}}(\mathbf{x}) = \frac{\mathcal{K}_\lambda(\mathbf{x}, \xi_{ij})}{\sum_{m=1}^{11} \sum_{l=1}^{11} \mathcal{K}_\lambda(\mathbf{x}, \xi_{lm})}$$

Make contour plots for $N_{.1,6,6}(\mathbf{x})$ and $N_{.01,6,6}(\mathbf{x})$.

2. (6HW-13) Consider again the data of Problem 1 of Section A.18. Fit (training-set-dependent) radial basis function networks based on the standard normal pdf ϕ ,

$$f_\lambda(x) = \beta_0 + \sum_{i=1}^{101} \beta_i \mathcal{K}_\lambda(x, x_i) \text{ for } \mathcal{K}_\lambda(x, x_i) = \phi\left(\frac{\|x - x_i\|}{\lambda}\right)$$

to these data for two different values of λ . Then define normalized versions of the radial basis functions as

$$\mathcal{N}_{\lambda_i}(x) = \frac{\mathcal{K}_\lambda(x, x_i)}{\sum_{m=1}^{101} \mathcal{K}_\lambda(x, x_m)}$$

and redo the fitting using the normalized versions of the basis functions.

3. (5HW-14) Fit radial basis function networks based on the standard normal pdf ϕ ,

$$f_{\lambda}(\mathbf{x}) = \beta_0 + \sum_{i=1}^{81} \beta_i \mathcal{K}_{\lambda}(\mathbf{x}, \mathbf{x}_i) \text{ for } \mathcal{K}_{\lambda}(\mathbf{x}, \mathbf{x}_i) = \phi\left(\frac{\|\mathbf{x} - \mathbf{x}_i\|}{\lambda}\right)$$

to the data of Problem 2 Section A.16 for two different values of λ . Then define normalized versions of the radial basis functions as

$$\mathcal{N}_{\lambda i}(\mathbf{x}) = \frac{\mathcal{K}_{\lambda}(\mathbf{x}, \mathbf{x}_i)}{\sum_{m=1}^{81} \mathcal{K}_{\lambda}(\mathbf{x}, \mathbf{x}_m)}$$

and redo the fitting using the normalized versions of the basis functions.

4. (6HW-15) Fit radial basis function networks based on the standard normal pdf ϕ ,

$$f_{\lambda}(x) = \beta_0 + \sum_{m=1}^{51} \beta_m \mathcal{K}_{\lambda}\left(x, \frac{m-1}{50}\right) \text{ for } \mathcal{K}_{\lambda}(x, z) = \phi\left(\frac{|x-z|}{\lambda}\right)$$

to the dataset of Problem 17 of Section A.2 for two different fixed values of λ . Define normalized versions of the radial basis functions as

$$\mathcal{N}_{\lambda i}(x) = \frac{\mathcal{K}_{\lambda}\left(x, \frac{i-1}{50}\right)}{\sum_{m=1}^{51} \mathcal{K}_{\lambda}\left(x, \frac{m-1}{50}\right)}$$

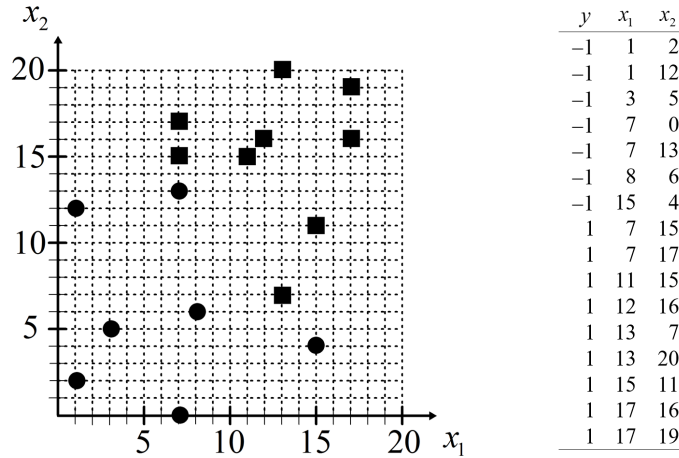
and redo the fitting using the normalized versions of the basis functions.

A.23 Section 9.1 Exercises

1. (5E1-18) Use the training set in Problem 3 of Section A.2 without bothering to center y , carefully build a binary regression tree with 6 final nodes (employing 5 splits, each at one of the values .2, .35, .5, .65, and .8). For each split, give the associated SSE provided by the split. Make a tree diagram for representing your development. If SSE is penalized by $\lambda = 6$ times the number of tree nodes, which of the trees met in your construction is most attractive?

2. (6E2-11) Below is a small $p = 2$ classification training set (for $K = 2$ classes) displayed in graphical and tabular forms (circles are class -1 and squares are

class 1).



a) Using empirical misclassification rate as your splitting criterion and standard forward selection, find a reasonably simple binary tree classifier that has training error rate 0. Provide the tree diagram and sketch the corresponding rectangles on a plot like the one above.

b) For every sub-tree, T , of your full binary tree above, find the size (number of final nodes) of the sub-tree, $|T|$, and the empirical error rate of its associated classifier.

c) Using the values from b), find for every $\alpha \geq 0$ a sub-tree of your full tree minimizing

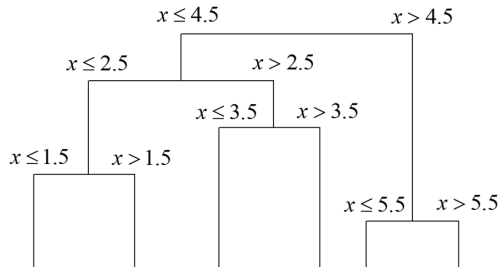
$$C_\alpha = |T| + \alpha \cdot \overline{\text{err}}$$

3. (6E1-13) Consider the $p = 1$ prediction problem with $N = 6$ and training data as below.

y	1.6	.4	3.5	1.5	5	6
x	1	2	3	4	5	6

Forward selection of binary trees for SEL prediction produces the sequence of trees represented below. If one determines to prune back from the final tree in optimal fashion, there is a nested sequence of subtrees that are the only possible optimizers of $C_\alpha(T) = |T| + \alpha \text{SSE}(T)$ for positive α . Identify that

nested sequence of sub-trees of Tree 5 below.



Tree Number	Subsets of values of x	SSE
0	1 2 3 4 5 6	24.22
1	1 2 3 4 5 6	5.47
2	1 2 3 4 5 6	3.22
3	1 2 3 4 5 6	1.22
4	1 2 3 4 5 6	.50
5	1 2 3 4 5 6	0

4. (5HW-14) Your instructor will provide an $N = 200$ training set generated from the "Friedman1" benchmark model (using the `mlbench` package in `R`. For purposes of assessing test error, you will also be given a size 5000 test set generated from this model.

a) Fit a single regression tree to the dataset. Prune the tree to get the best sub-trees of all sizes from 1 final node to the maximum number of nodes produced by the tree routine. Compute and plot cost-complexities $C_\alpha(T)$ as a function of α .

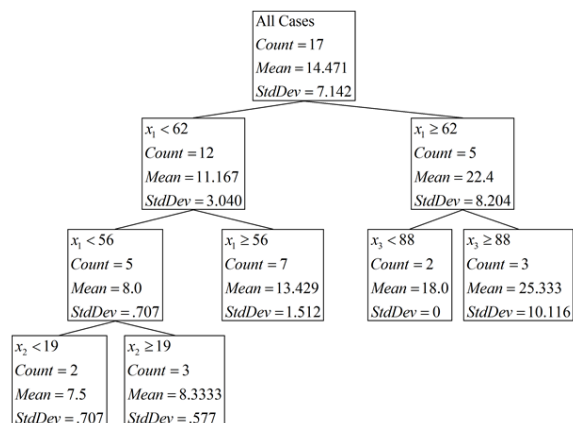
b) Evaluate a "test error" (based on the size 5000 test set) for each sub-tree identified in **a)**. What size sub-tree looks best based on these values?

c) Do 5-fold cross-validation on single regression trees to pick an appropriate tree complexity (to pick one of the sub-trees from **a)**). How does that choice compare to what you got in **b)** based on test error for the large test set?

5. (5HW-14) Return to the context of Problem 7 Section A.2 and Problem 1 Section A.5. Fit a classification tree to the dataset using 5-fold cross-validation to choose tree size based on cost-complexity tuning. Make a plot like that required in the earlier problems showing the regions where the tree classifies to each of the 4 classes. Evaluate the (conditional on the training set) test error rate for this tree.

6. (5E1-16) Below is a representation of a binary regression tree. Find a subtree of this tree that minimizes the cost $C_\alpha(T) = |T| + \alpha \text{SSE}(T)$ for $\alpha = .01$. (There are 7 subtrees to consider.) Identify the final nodes for the optimal

subtree.



7. (5E2-14) Consider a context in which for a continuous input $x \in [0, 1]$ the conditional mean function $E[y|x]$ is strictly increasing. Argue carefully that any binary tree predictor must be positively biased at $x = 0$ and negatively biased at $x = 1$ in this context.

A.24 Section 10.1 Exercises

1. (5E1-18) Use the training set in Problem 3 of Section A.2 and without bothering to center y , consider bagging a SEL predictor for y of the form

$$\hat{f}(x) = b_1 I[x < .5] + b_2 I[x \geq .5]$$

fit by OLS. Below, $B = 10$ bootstrap samples are represented in terms of case indices and the corresponding values of b_1 and b_2 are provided. Find an OOB MSPE for a bagged predictor $\hat{f}_{\text{bag}}^{10}$.

Bootstrap Sample	b_1	b_2
2, 3, 5, 5, 5, 6	7.000	7.000
2, 2, 3, 4, 5, 6	6.000	9.333
1, 1, 1, 1, 2, 6	.8000	10.000
1, 1, 3, 4, 5, 6	3.333	9.333
1, 1, 2, 3, 5, 6	3.500	8.000
1, 1, 2, 4, 4, 5	1.333	10.000
2, 2, 2, 3, 5, 5	4.000	6.000
2, 3, 3, 4, 4, 5	8.000	10.000
1, 2, 2, 2, 3, 4	3.200	12.000
2, 3, 4, 5, 6, 6	7.000	8.666

2. (6E1-19) All cases in a particular $N = 100$ training set are distinct/different. Suppose that one is going to make "weighted bootstrap samples" of size 100, using not equal weights of .01 on each case in the training set, but rather weights/probabilities w_1, w_2, \dots, w_{100} (where each $w_i > 0$ and $\sum_{i=1}^{100} w_i = 1$).

a) What is the probability that a case with $w_i = .02$ is included in a particular weighted-bootstrap sample of size 100?

b) Suppose that for $b = 1, 2, \dots, B$ the corresponding weighted bootstrap sample is \mathbf{T}_b^* and the sample mean of responses in this sample is \bar{y}^{*b} . Further, let $\bar{y}_{\text{bag}} = \frac{1}{B} \sum_{b=1}^B \bar{y}^{*b}$. Find an expression for $\lim_{B \rightarrow \infty} \bar{y}_{\text{bag}}$ and argue carefully that your expression is correct.

A.25 Section 10.2 Exercises

1. (6E1-13) Consider a $p = 1$ prediction problem for $x \in [0, 1]$ and random forest predictor \hat{f}_B^* based on a training set of size $N = 101$ with $x_i = (i - 1) / 100$ for $i = 1, \dots, 101$ and $n_{\min} = 5$ (so no split is made in creating a single tree predictor \hat{f}^{*b} that would produce a leaf representing fewer than 5 training points).

a) Use simulation to approximate the expected value of the arithmetic mean of the 5 largest of 101 values drawn at random with replacement from $\{.00, .01, \dots, 1.00\}$. Call this value η .

b) Consider the bias of prediction at $x = 1.00$, namely

$$E\left(\hat{f}_B^*(1.00) - 1.00\right)$$

under a model where $E y_i = x_i$. Use your value η from a) to argue carefully that this bias is clearly negative.

2. (5HW-14) Return to the context of Problem 4 of Section A.23.

a) Use the `randomForest` package in R to make a bagged version of a regression tree (based on, say, $B = 500$). What is its OOB error? How does that compare to a test error based on the size 5000 test set?

b) Make a random forest predictor using `randomForest` (use again $B = 500$). What is its OOB error? How does that compare to a test error based on the size 5000 test set?

3. (6E1-17) If, in a classification problem, all N inputs $\mathbf{x}_i \in \mathfrak{R}^p$ are distinct, a default random forest (one with $n_{\min} = 1$) will typically have $\overline{\text{err}} = 0$ (a 0 training error rate for 0-1 loss) unless a "small" maximum tree depth is set.

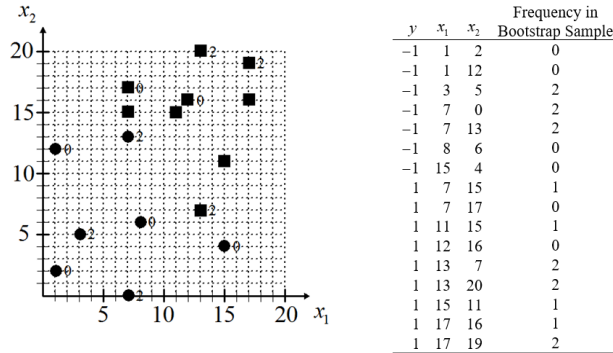
a) Why is this? Explain.

b) Does this mean that the OOB error rate will be 0? Explain.

c) Does this mean that the OOB error rate is unreliable as a representing likely random forest performance? Explain.

4. (6E2-15) Below is a small $p = 2$ classification training set (for 2 classes) displayed in graphical and tabular forms (circles are class -1 and squares are

class 1). A bootstrap sample is made from this dataset and is indicated in the table *and by counts next to plotted points for those points represented in the sample other than once*. This sample is used to create a tree in a random forest with 4 end nodes (accomplished by 3 binary splits). A random choice is made for which of the 2 variables to split on at each opportunity and turns out to produce the sequence " x_1 then x_1 then x_2 ."



- a) Identify the resulting tree by rectangles on the plot and provide the value of \hat{y} for each rectangle.
- b) Which out-of-bag points are misclassified by this particular tree?

5. (5E1-16) A variant of the random forest algorithm begins by making a random p -dimensional rotation of the predictors of a bootstrap sample before building the tree for that bootstrap sample, \hat{f}^{*b} . (You may, for example, think of this in terms of the $p = 2$ case for inputs \mathbf{x} , and rotating the 2-d coordinate axes around the origin before doing splitting based on the 2 rotated axes.) What about this innovation is *attractive* and what about it is *unattractive*?

A.26 Section 11.1 Exercises

1. (6HW-11) Below is a very small sample of fictitious $p = 1$ training data.

x	1	2	3	4	5
y	1	4	3	5	6

Consider a toy Bayesian model averaging problem where what is of interest is a prediction for y at $x = 3$. Suppose that under Model 1, the (x_i, y_i) are iid where x is Discrete Uniform on $\{1, 2, 3, 4, 5\}$ and $y|x$ is Binomial(10, $p(x)$) for $p(x) = \Phi\left(\frac{x-a}{b}\right)$ (for Φ the standard normal cdf and $b > 0$). In this model, the quantity of interest is $10 \cdot \Phi\left(\frac{3-a}{b}\right)$.

On the other hand, suppose that under Model 2, the (x_i, y_i) are iid where x is Discrete Uniform on $\{1, 2, 3, 4, 5\}$ and $y|x$ is Binomial(10, $p(x)$) for $p(x) = 1 - \frac{1}{(c+1)^x}$ (for some $c > 0$). In this model, the quantity of interest is $10 \cdot \left(1 - \frac{1}{3(c+1)}\right)$.

For prior distributions, suppose that for Model 1 a and b are *a priori* independent with $a \sim U(0, 6)$ and $b^{-1} \sim \text{Exp}(1)$, while for Model 2, $c \sim \text{Exp}(1)$. Further suppose that prior probabilities on the models are $\pi(1) = \pi(2) = .5$. Compute (almost surely you'll have to do this numerically) posterior means of the quantities of interest in the two Bayes models, posterior probabilities for the two models, and the overall predictor of y at $x = 3$.

2. (6E2-11) Consider a Bayesian model averaging problem where x takes values in $\{0, 1\}$ and y takes values in $\{0, 1\}$. The quantity of interest is

$$P[y = 1|x = 1] / P[y = 0|x = 1]$$

and there are $M = 2$ models under consideration. We'll suppose that joint probabilities for (x, y) are as given in the tables below for the two models for some $p \in (0, 1)$ and $r \in (0, 1)$

Model 1			Model 2		
$y \setminus x$	0	1	$y \setminus x$	0	1
1	.25	.25	1	$(1-r)/2$	$r/2$
0	$(1-p)/2$	$p/2$	0	.25	.25

so that under Model 1, the quantity of interest is $.5/p$ and under Model 2, it is $r/.5$. Suppose that under both models, training data (x_i, y_i) for $i = 1, \dots, N$ are iid. For priors, suppose that in Model 1 *a priori* $p \sim \text{Beta}(2, 2)$ and suppose that in Model 2 *a priori* $r \sim \text{Beta}(2, 2)$. Further, suppose that the prior probabilities of the two models are $\pi(1) = \pi(2) = .5$.

Find the posterior probabilities of the 2 models, $\pi(1|\mathbf{T})$ and $\pi(2|\mathbf{T})$ and the Bayes model average squared error loss predictor of $P[y = 1|x = 1] / P[y = 0|x = 1]$. (You may think of the training data as summarized in the 4 counts $N_{(x,y)}$ = number of training vectors with value (x, y) .)

3. (6E1-13) Consider a simple Bayes model averaging prediction problem with iid training data (x_i, y_i) where $x_i \in \{0, 1\}$ and we assume that $y_i = \mu(x_i) + \varepsilon_i$ for $\varepsilon_i \sim N(0, 1)$. Two models are contemplated. Model 1 says that $\mu(0) = \mu(1) = \mu$ and *a priori* $\mu \sim N(0, (10)^2)$. Model 2 says that $\mu(0)$ and $\mu(1)$ are *a priori* independent with both $\mu(0) \sim N(0, (10)^2)$ and $\mu(1) \sim N(0, (10)^2)$. Assume that *a priori* the two models are equally likely. Training pairs (x_i, y_i) are $(0, 5), (0, 7), (0, 6), (1, 12)$. Find an appropriate predicted value of y if $x = 1$.

You will find likely it helpful to recall that if conditioned on θ , observations z_1, \dots, z_n are iid $N(\theta, 1)$ and θ is itself $N(0, \tau^2)$, then conditioned on z_1, \dots, z_n , θ is $N\left(\left(\frac{n}{n+\frac{1}{\tau^2}}\right) \bar{z}, \left(n + \frac{1}{\tau^2}\right)^{-1}\right)$

4. (6E1-15) Below are tables specifying two discrete joint distributions for (x, y) that we'll call Model 1 and Model 2. Suppose that $N = 2$ training cases

(drawn iid from one of the models) are $(x_1, y_1) = (2, 2)$ and $(x_2, y_2) = (3, 3)$.

Model 1				Model 2			
$y \setminus x$	1	2	3	$y \setminus x$	1	2	3
3	0	.125	.125	3	0	0	.1
2	0	.125	.125	2	.1	.2	.1
1	.125	.125	0	1	.1	.2	.1
0	.125	.125	0	0	.1	0	0

Suppose further that prior probabilities for the two models are $\pi_1 = .3$ and $\pi_2 = .7$.

a) Find the posterior probabilities of Models 1 and 2.

b) Find the "Bayes model averaging" SEL predictor of y based on x for these training data. (Give values $\hat{f}(1)$, $\hat{f}(2)$, and $\hat{f}(3)$.)

A.27 Section 11.2 Exercises

1. (6E2-11) The machine learning/data mining folklore is full of statements like "combining uncorrelated classifiers through majority voting produces a committee classifier better than every individual in the committee." This is simply not necessarily true. Consider the Vardeman and Morris scenario outlined in the table below as regards the joint distribution of classifiers f_1, f_2 , and f_3 and a target (class variable) y taking values in $\{0, 1\}$.

Outcome	f_1	f_2	f_3	y	Probability
1	0	0	0	0	0
2	0	0	1	0	.008
3	0	1	0	0	.008
4	1	0	0	0	.008
5	0	1	1	0	.08
6	1	0	1	0	.08
7	1	1	0	0	.08
8	1	1	1	0	0
9	0	0	0	1	$(.1)^3$
10	0	0	1	1	$(.9)(.1)^2 - .008$
11	0	1	0	1	$(.9)(.1)^2 - .008$
12	1	0	0	1	$(.9)(.1)^2 - .008$
13	0	1	1	1	$(.9)^2(.1) - .08$
14	1	0	1	1	$(.9)^2(.1) - .08$
15	1	1	0	1	$(.9)^2(.1) - .08$
16	1	1	1	1	$(.9)^3$

a) Find the expected 0-1 loss for the individual classifiers and for the "majority vote" classifier. Note that the classifiers are independent according to this joint distribution.

b) Treat the vector of values of $f_1, f_2,$ and f_3 as "available data" and find the conditional distributions of the vector given $y = 0$ and $y = 1$. What is in fact the best function of these classifiers in terms of expected 0-1 loss? (Look again at Sections 1.4 and 1.5.) How does its error rate compare to the error rates from a)?

A.28 Section 11.4 Exercises

1. (5E1-18) Again use the training set in Problem 3 of Section A.2 without bothering to center y , consider using boosting to create a SEL predictor for it. As your set of "basis functions for successive corrections" adopt the 10 indicator functions

$$l_1(x) = I[x < .2], l_2(x) = I[x < .35], l_3(x) = I[x < .5], l_4(x) = I[x < .65], \\ l_5(x) = I[x < .8], u_1(x) = I[x \geq .2], u_2(x) = I[x \geq .35], u_3(x) = I[x \geq .5], \\ u_4(x) = I[x \geq .65], u_5(x) = I[x \geq .8]$$

Take $\hat{f}_0(x) = \bar{y}$ and using a "learning rate" of .5, find $\hat{f}_1(x)$, the first boosted iterate. (This will be $\hat{f}_0(x)$ plus a multiple of one of the indicator functions.)

2. (6HW-11) (Izenman Problem 14.4.) Consider 2-class classification problem with input space \mathfrak{R}^2 and $N = 10$ observations in the table below.

y	1	1	1	1	1	2	2	2	2	2
x_1	1	3.5	4.5	6	1.5	8	3	4.5	8	2.5
x_2	4	6.5	7.5	6	1.5	6.5	4.5	4	1.5	0

Plot the (x_1, x_2) pairs on a scatterplot using different symbols or colors to distinguish the two classes (1 and 2). Carry through the AdaBoost.M1 algorithm on these points "by hand" for $M = 4$ iterations, showing the weights at each step of the process. Determine the voting function and final classifier and calculate its training error rate.

3. (6E2-11) Find the $M = 3$ AdaBoost.M1 classifier for the data of Problem 2 of Section A.23.

4. (6HW-13) Consider the famous Swiss Bank Note dataset. Use `caret train()` to choose (via LOOCV) both AdaBoost.M1 and random forest 0-1 loss classifiers based on these data. For a fine grid of points indicate on a 2-d plot which points get classified to classes -1 and 1 so that you can make visual comparisons.

5. (6HW-13) This problem concerns the "Seeds" dataset at the UCI Machine Learning Repository. Standardize all $p = 7$ input variables before beginning analysis.

a) Consider first the problem of classification where only varieties 1 and 3 are considered (temporarily code variety 1 as -1 and variety 3 as $+1$) and use

only predictors x_1 and x_6 . Use `caret train()` to choose (via LOOCV) both AdaBoost.M1 and random forest 0-1 loss classifiers based on these data. For a fine grid of points in $[-3, 3] \times [-3, 3]$, indicate on a 2-d plot which points get classified to classes -1 and 1 so that you can make visual comparisons.

b) The paper "ada: An R Package for Stochastic Boosting" by Culp, Johnson, and Michailidis that appeared in the *Journal of Statistical Software* discusses using a one-versus-all strategy to move AdaBoost to a multi-class problem known as the "AdaBoost.MH" algorithm. Continue the use of only predictors x_1 and x_6 and find both an appropriate random forest classifier and an AdaBoost.MH classifier for the 3-class problem with $p = 2$, and once more show how the classifiers break the 2-d input space up into regions of constant classification.

c) How much better can you do at the classification task using a random forest classifier based on all $p = 7$ input variables than you are able to do in part **b)**? (Use LOOCV error rate to make your comparison.)

6. (6E2-13) Below is a toy $K = 2$ class training set for $N = 4$. Carry out ("by hand") enough steps of the AdaBoost.M1 algorithm (find a number of iterations M large enough) to produce a voting function with 0 training error rate. Plot this function and indicate on the x axis which regions call for classification to the $y = 1$ class.

y	1	-1	1	-1
x	1	2	3	5

7. (5HW-14) Return to the context of Problem 4 of Section A.23.

a) Use the `gbm` package in R to fit several boosted regression trees to the training set (use at least 2 different values of tree depth with at least 2 different values of learning rate). What are values of training error and then test error based on the size 5000 test set for these?

b) How do predictors in Problem 4 of Section A.23, Problem 2 of Section A.25, and here compare in terms of test error? Evaluate \hat{y} for each of the first 5 cases in your test set for all predictors and list all of the inputs and each of the predictions in a small table.

c) Call your predictor from Problem 2 of Section A.25 \hat{f}_1 and pick one of your predictors from **a)** to call \hat{f}_2 . Use your test set and approximate

$$E\left(y - \hat{f}_1(x)\right), E\left(y - \hat{f}_2(x)\right), \text{Var}\left(y - \hat{f}_1(x)\right), \text{Var}\left(y - \hat{f}_2(x)\right),$$

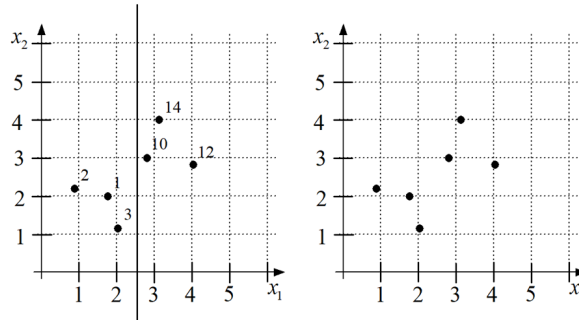
and $\text{Corr}\left(y - \hat{f}_1(x), y - \hat{f}_2(x)\right)$

(these expectations are across the joint distribution of (x, y) for the fixed training set (and randomization for the random forest). Identify an α approximately optimizing

$$E\left(y - \left(\alpha \hat{f}_1(x) + (1 - \alpha) \hat{f}_2(x)\right)\right)^2$$

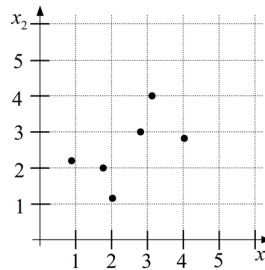
Is the optimizer 0 or 1 (i.e. is the best linear combination of the two predictors one of them alone)?

8. (6E1-19) Below is a toy $p = 2$ training set with $N = 6$. (The 6 values of y are plotted near $\mathbf{x} = (x_1, x_2)$ locations corresponding to their input vectors.) Consider SEL boosting using "2-split SEL regression trees" (trees with 3 final nodes) as base predictors. (Two splits are made to produce each $\hat{e}_m(\mathbf{x})$.)



a) Beginning with $\hat{f}_0(\mathbf{x}) \equiv 7$ and the first split of iteration 1 (for making $\hat{e}_1(\mathbf{x})$) as indicated on the left figure, draw in the 2nd split. Using it and a $\nu = .5$ learning rate, place the $N = 6$ values $y_i - \hat{f}_1(\mathbf{x}_i)$ onto the right figure. On that, mark the 2 cuts for creating $\hat{e}_2(\mathbf{x})$.

b) Then, again using a $\nu = .5$ learning rate and now your $\hat{e}_2(\mathbf{x})$ implied by the 2 cuts on the right figure above, below show the regions on which $\hat{f}_2(\mathbf{x})$ is constant and indicate the values of $\hat{f}_2(\mathbf{x})$ in those regions.



9. (6E1-17) Suppose that in a toy 2-class classification model with $p = 1$ using the $y \in \{-1, 1\}$ coding one has $N = 5$ training cases in the small table below.

y	-1	-1	1	1	-1
x	-1.5	-0.5	0.5	1.5	2.5

In a gradient boosting exercise with the hinge loss

$$\sum_{i=1}^5 (1 - y_i g(x_i))_+$$

and base functions $I[x < c]$ and $I[x > c] \forall c$, suppose that one has a current function version $g_m(x) = 3x$. Derive the function $g_{m+1}(x)$.

10. (6E1-15) Consider the $p = 2$ prediction problem based on $N = 9$ training points as below.

$$\mathbf{Y} = \frac{1}{\sqrt{6}} \begin{pmatrix} 8 \\ 3 \\ -3 \\ 5 \\ -1 \\ -5 \\ 1 \\ -3 \\ -5 \end{pmatrix} \text{ and } \mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2) = \frac{1}{\sqrt{6}} \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 1 & -1 \\ 0 & 1 \\ 0 & 0 \\ 0 & -1 \\ -1 & 1 \\ -1 & 0 \\ -1 & -1 \end{pmatrix}$$

- a) Find the SEL lasso coefficient vector $\hat{\beta}$ optimizing $\text{SSE} + 8 \left(\left| \hat{\beta}_1^{\text{lasso}} \right| + \left| \hat{\beta}_2^{\text{lasso}} \right| \right)$ and give the corresponding $\hat{\mathbf{Y}}^{\text{lasso}}$.
- b) "Boost" your lasso SEL predictor from a) using ridge regression with $\lambda = 1$ and a learning rate of $\nu = .1$. Give the resulting vector of predictions $\hat{\mathbf{Y}}^{\text{boost1}}$.
- c) Why is it clear that the predictor in b) is a linear predictor? What is $\hat{\beta}$ such that $\hat{\mathbf{Y}}^{\text{boost1}} = \mathbf{X}\hat{\beta}$?
- d) Now "boost" your SEL lasso predictor from a) using a best "stump" regression tree predictor (one that makes only a single split) and a learning rate of $\nu = .1$. Give the resulting vector of predictions $\hat{\mathbf{Y}}^{\text{boost2}}$.

11. (6E2-15) The AdaBoostM.1 classification algorithm is essentially an application of general gradient boosting to exponential loss and basic function updates that are simple "binary stumps." This problem concerns applying the algorithm with hinge loss, $L(\hat{y}, y) = [1 - y\hat{y}]_+$ (for the -1 and 1 coding for y and $\hat{y} \in \mathcal{R}$), and *linear functions* of predictor $\mathbf{x} \in \mathcal{R}^p$, say $\beta_0 + \mathbf{x}'\beta$, as basic function updates. (Of course, since linear combinations of linear functions are linear, this can only produce a best linear voting function.)

- a) What starting function $f_0(\mathbf{x})$ would be used?
- b) With the $(m - 1)$ iterate $f_{m-1}(\mathbf{x})$ in hand, each \tilde{y}_{im} is in $\{-1, 1\}$. Using appropriate indicator functions, give an explicit formula for \tilde{y}_{im} in terms of y_i and $\hat{y}_{im-1} = f_{m-1}(\mathbf{x}_i)$.
- c) Describe in words how you would use standard statistical software to produce β_{0m} and β_m so that all $\beta_{0m} + \mathbf{x}'_i\beta_m$ approximate the values \tilde{y}_{im} .
- d) Why does optimization of $\sum_{i=1}^N [1 - y_i (f_{m-1}(\mathbf{x}_i) + \rho(\beta_{0m} + \mathbf{x}'_i\beta_m))]_+$ over choices of ρ involve comparison of this quantity for at most N values of ρ ? Give a formula for values of ρ that you might have to check.

e) After M iterations you won't have an $f_M(\mathbf{x})$ taking only values -1 and 1 at every \mathbf{x}_i . How do you use $f_M(\mathbf{x})$ to do classification?

12. (5HW-16) Use R and make a simple set of boosted predictions of home price for the dataset of Problem 5 Section A.2 by first fitting a "default" random forest (using `randomForest`), then correcting a fraction $\nu = .1$ of the residuals predicted using a 7-nn predictor, then correcting a fraction $\nu = .1$ of the residuals predicted using a 1 component PLS predictor. Then permute the orders in which you make these corrections and compare SSE for the 6 different possibilities.

13. (5E2-14) Below are hypothetical counts from a small training set in a 2-class classification problem with a single input, $x \in \mathfrak{R}$ (and we'll treat x as integer-valued). Although it is easy to determine what an approximately optimal (0-1 loss) classifier is here, instead consider use of the AdaBoost.M1 algorithm to produce a classifier. (Use "stumps"/two-node trees that split *between integer values* as basis functions.) Find an $M = 3$ term version of the AdaBoost.M1 voting function. (Give $\hat{f}_1, \alpha_1, \hat{f}_2, \alpha_2, \hat{f}_3,$ and α_3 . The \hat{f}_m s are of the form $\text{sign}(x - \#)$ or $\text{sign}(\# - x)$ and the final voting function is $\sum_{m=1}^3 \alpha_m \hat{f}_m$.)

	$x = 1$	$x = 2$	$x = 3$
$y = 1$	3	5	2
$y = -1$	5	4	6

14. (5E1-20) In a toy $p = 1$ SEL prediction problem, the table in Problem 28 of the Section A.2 provides $N = 5$ training cases. An initial predictor $\hat{f}_0(x) \equiv 0$ is boosted using simple linear regression and a learning rate of $\nu = 1/3$ to produce the predictor $\hat{f}_1(x) = .5x$. This problem is about making 2 more "boosting" steps to produce $\hat{f}_3(x)$.

a) Make a 1-nn "boosting" correction to $\hat{f}_1(x)$ with learning rate $\nu = .5$ to produce $\hat{f}_2(x) = \hat{f}_1(x) + .5\hat{e}_2(x)$. (Give a formula/expression for $\hat{e}_2(x)$, a step function constant on 5 consecutive intervals.)

b) Find values for $\hat{f}_2(x)$ at $x = -2, -1, 0, 1, 2$. Then consider a "regression tree with a single split" "boosting" correction to this predictor. Choose from values $\{-1.5, -.5, .5, 1.5\}$ for the location of your split (justifying your choice) and then give a formula for $\hat{e}_3(x)$ (a step function taking 2 values).

A.29 Section 12.1 Exercises

1. (6HW-11) Figure 4.4 of HTF gives a 2-dimensional plot of the "vowel training data" (available on the book's website at <http://www-stat.stanford.edu/~tibs/ElemStatLearn/index.html> or from the UCI data repository. The ordered pairs of first 2 canonical variates are plotted to give a "best" reduced rank LDA picture of the data like that below (lacking the decision boundaries).

Use the material of Section 12.1 to reproduce Figure 4.4 of HTF (color-coded by group, with group means clearly indicated). Keep in mind that you may need to multiply one or both of your coordinates by -1 to get the exact picture.

2. (6E2-11) Suppose that in a $p = 2$ linear discriminant analysis problem, four transformed means $\boldsymbol{\mu}_k^* = \boldsymbol{\Sigma}^{-\frac{1}{2}}(\boldsymbol{\mu}_k - \bar{\boldsymbol{\mu}})$ are $\boldsymbol{\mu}_1^* = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, $\boldsymbol{\mu}_2^* = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$, $\boldsymbol{\mu}_3^* = \begin{pmatrix} 3.5 \\ 1.5 \end{pmatrix}$, and $\boldsymbol{\mu}_4^* = \begin{pmatrix} .5 \\ 2.5 \end{pmatrix}$. These have sample covariance matrix

$$\begin{pmatrix} 3.125 & 1.625 \\ 1.625 & 3.125 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \mathbf{diag}(4.75, 1.5) \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$$

Suppose that one wants to do reduced rank ($rank = 1$) linear discrimination based on a single real variable

$$w = (u_1, u_2) \boldsymbol{\Sigma}^{-\frac{1}{2}}(\mathbf{x} - \bar{\boldsymbol{\mu}})$$

Identify an appropriate vector (u_1, u_2) and with your choice of vector, give the function $f(w)$ mapping $\mathfrak{R} \rightarrow \{1, 2, 3, 4\}$ that defines this 4-class classifier for the case of $\pi_1 = \pi_2 = \pi_3 = \pi_4$.

3. (6E2-13) In a 6-class, $p = 3$ linear discriminant problem with equal class probabilities ($\pi_1 = \pi_2 = \pi_3 = \pi_4 = \pi_5 = \pi_6$), unit eigenvectors corresponding to the largest 2 eigenvalues of the sample covariance matrix of the sphered (according to the common within-class covariance matrix) class means are respectively

$$\mathbf{v}_1 = \left(\frac{1}{\sqrt{2}}, 0, -\frac{1}{\sqrt{2}} \right)' \text{ and } \mathbf{v}_2 = (0, 1, 0)'$$

Suppose that inner product pairs $(\langle \boldsymbol{\mu}_k^*, \mathbf{v}_1 \rangle, \langle \boldsymbol{\mu}_k^*, \mathbf{v}_2 \rangle)$ for the sphered class means are as below and that reduced rank ($rank = 2$) linear classification is of interest. How should a sphered $p = 3$ observation $\mathbf{x}^* = (3, 4, -5)'$ be classified?

Class	1	2	3	4	5	6
Inner Product Pair	(5, 0)	(-5, 0)	(0, 3)	(0, -3)	(0, 0)	(0, 0)

4. (6HW-17) Use the "Glass Identification" dataset referred to in Problem 6 of Section A.2. Do the following with it, not using the "problematic-looking" inputs "Ba" and "Fe".

a) Use the `lda()` function in the MASS package and do LDA based on all $p = 7$ inputs and find LOOCV 0-1 loss error rates for each type of glass and overall.

b) Using the function `stepclass()` in the R package `klaR` (or otherwise) use cross-validation to select a number of variables to use in linear discriminant

analysis for classification among the 6 glass types. Then choose this number of input variables by forward selection with the whole dataset. What are they?

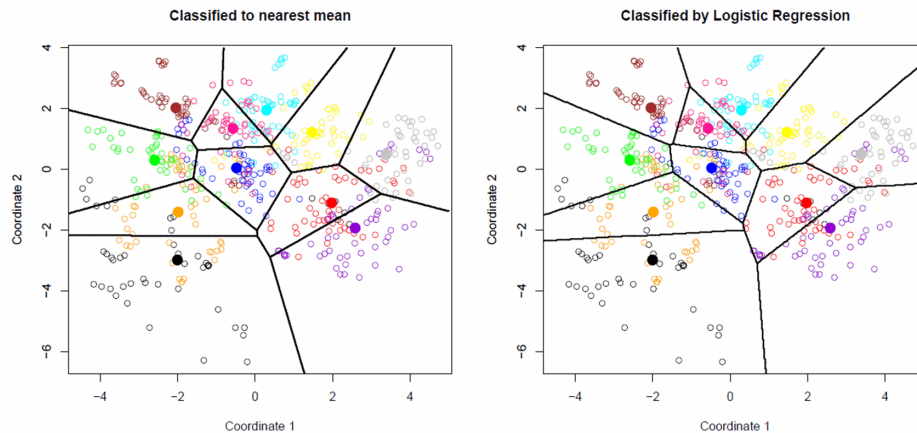
c) Find the first 2 canonical coordinates for all 215 cases in the dataset. Plot $N = 215$ ordered pairs of these using different plotting symbols for the $K = 6$ glass types. Overlay on this plot classification regions based on LDA with these first 2 canonical coordinates. Make a plot analogous to the plot in Figure 4.11 of HTF. (You may simply differently color points on a fine grid according to which glass such a point would be classified to.)

A.30 Section 12.2 Exercises

1. (6HW-11) Consider again the context of Problem 1 of Section A.29.

a) Use the R function `lda` (in the MASS package) to obtain the group means and coefficients of linear discriminants for the vowel training data. Save the `lda` object by a command such as `LDA=lda(insert formula, data=vowel)`.

b) Reproduce a version of the left figure below. You will need to plot the first two canonical coordinates as in Problem 1 of Section A.29. Decision boundaries for this figure are determined by classifying to the nearest group mean. Do the classification for a fine grid of points covering the entire area of the plot. You may plot the points of the grid with color coding according to their classification instead of drawing in the black lines.



c) Make a version of the right figure above with decision boundaries now determined by using logistic regression as applied to the first two canonical variates. You will need to create a data frame with columns y , *canonical variate 1*, and *canonical variate 2*. Use the `vglm` function (in the VGAM package) with `family=multinomial()` to do the logistic regression. Save the object created by a command such as `LR=vglm(insert formula, family=multinomial(), data=data set)`. A set of observations can now be classified to groups by using the command `predict(LR, newdata, type="response")`, where `newdata`

contains the observations to be classified. The outcome of the `predict` function will be a matrix of probabilities. Each row contains the probabilities that a corresponding observation belongs to each of the groups (and thus sums to 1). We classify to the group with maximum probability. As in **b**), do the classification for a fine grid of points covering the entire area of the plot. You may again plot the points of the grid, color-coded according to their classification, instead of drawing in the black lines.

d) So that you can plot results, first use the 2 canonical variates employed thus far and use `rpart` in **R** to find a classification tree with training error rate comparable to the reduced rank LDA classifier pictured on the left above. Make a plot showing the partition of the region into pieces associated with the 11 different classes. (The intention here is that you show rectangular regions indicating which classes are assigned to each rectangle, in a plot that might be compared to the plots above and from Problem 1 of Section A.29.)

e) The Culp, Johnson, and Michailidis paper referred to in Problem 5 of Section A.28 discusses using a one-versus-all strategy that moves AdaBoost to a multi-class problem known as the "AdaBoost.MH" algorithm. Continue the use of the first two canonical coordinates of the vowel training data and find both an appropriate random forest classifier and an AdaBoost.MH classifier for the 11-class problem with $p = 2$, and once more show how the classifiers break the 2-d space up into regions to be compared to other plots here.

f) Beginning with the original vowel dataset (rather than with the first 2 canonical variates) and use `rpart` in **R** to find a classification tree with training error rate comparable to the classifier in **d**). How much (if any) simpler/smaller is the tree here than in **d**)?

2. (6HW-13) Consider again the Swiss Bank Note dataset of Problem 4 of Section A.28. Use `caret train()` to choose (via LOOCV using `glmnet`) a logistic regression-based 0-1 loss classifier based on these data. Compare its training set and cross-validation error rates to what you found in Section A.28.

3. (5E2-14) Overall, only a very small fraction of people presented with a certain merchandising offer will respond to it. A set of 5 qualitative predictors (potential personal traits) is thought to be related to response. Values for these 5 predictors are obtained from a group of 96 people who responded to the offer and from a group of 604 who did not. Treating the input x_j as taking the value 1 if a subject has trait j and 0 otherwise, a model for

$$p(\mathbf{x}) = \text{probability of responding to the offer given characteristics } \mathbf{x}$$

of the form

$$\log\left(\frac{p(\mathbf{x})}{1-p(\mathbf{x})}\right) = \beta_0 + \beta_1x_1 + \beta_2x_2 + \beta_3x_3 + \beta_4x_4 + \beta_5x_5$$

was fit (via maximum likelihood) to the 700 training cases yielding results

$$\hat{\beta}_0 = -3.42, \hat{\beta}_1 = 0.41, \hat{\beta}_2 = 1.76, \hat{\beta}_3 = -0.03, \hat{\beta}_4 = 0.13, \hat{\beta}_5 = 2.09$$

a) Treating the 700 subjects (that were used to fit the logistic regression model) as a random sample of people of interest (which it is surely not) give a linear function $g(\mathbf{x})$ such that $\hat{f}(\mathbf{x}) = I[g(\mathbf{x}) > 0]$ is an approximately optimal (0-1 loss) classifier ($y = 1$ indicating response to the offer).

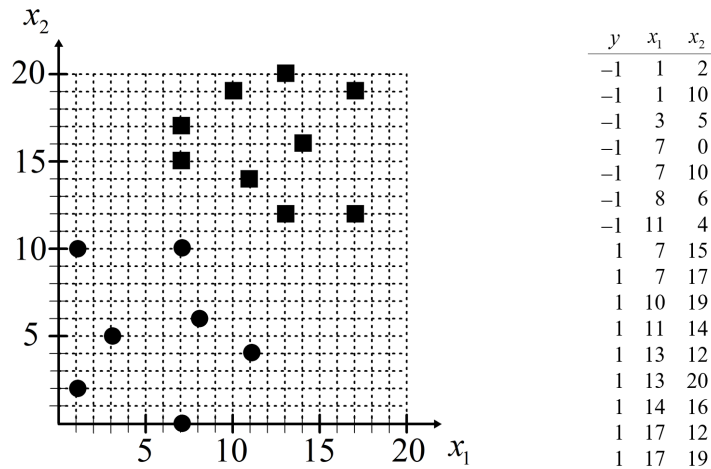
b) Continuing with the logistic regression model, properly adjust your answer to a) to provide an approximately optimal (0-1 loss) classifier for a case where a fraction $\pi_1 = 1/1000$ of all potential customers would respond to the offer.

A.31 Section 13.1 Exercises

1. (6E1-11) Below is a small classification training set (for $K = 2$ classes) displayed in graphical and tabular forms (circles are class -1 and squares are class 1). Using geometry (not trying to solve an optimization problem analytically) find the maximum margin classifier for this problem. You may find it helpful to know that if \mathbf{u}, \mathbf{v} , and \mathbf{w} points in \mathbb{R}^2 and $u_1 \neq v_1$ then the distance from the point \mathbf{w} to the line through \mathbf{u} and \mathbf{v} is

$$\frac{|w_1(v_2 - u_2) - w_2(v_1 - u_1) + v_1u_2 - u_1v_2|}{\sqrt{(v_1 - u_1)^2 + (v_2 - u_2)^2}}$$

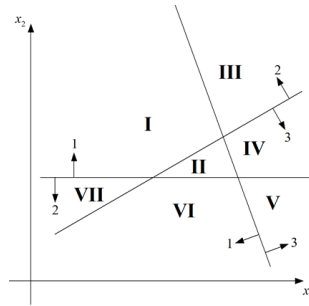
List the set of support vectors and evaluate the margin for your classifier.



A.32 Section 13.2 Exercises

1. (6HW-11) Consider again the Wisconsin breast cancer dataset of Problem 24 of Section A.2. Compare an appropriate support vector classifier (SVM with "linear kernel") based on the original input variables to a classifier based on logistic regression using the same variables. (Use `caret train()` to identify "best" versions of these classifiers in terms of LOOCV misclassification rates.)

2. (5E2-14) Below is a cartoon representing the results of 3 different runs of support vector classification software on a set of training data representing $K = 3$ different classes in a problem with input space \mathfrak{R}^2 . Each pair of classes was used to produce a linear classification boundary for classification between those two. (Labeled arrows tell which sides of the lines correspond to classification to which classes.) 7 different regions are identified by Roman numerals on the cartoon. Indicate values of an OVO (one-versus-one) classifier \hat{f}^{OVO} for this situation. (For each region, identify decisions 1, 2, or 3, or "?" if there is no clear choice for a given region.)



A.33 Section 13.3 Exercises

1. (6HW-11) This problem concerns the famous $p = 2$ "Ripley dataset" (`synth.tr`) commonly used as a classification example.

a) Using several different values of λ and constants c , find a function $g \in \mathcal{A}$ and $\beta_0 \in \mathfrak{R}$ minimizing

$$\sum_{i=1}^N (1 - y_i (\beta_0 + g(\mathbf{x}_i)))_+ + \lambda \|g\|_{\mathcal{A}}^2$$

for the (Gaussian) kernel $\mathcal{K}(\mathbf{x}, \mathbf{z}) = \exp(-c \|\mathbf{x} - \mathbf{z}\|^2)$. Make contour plots for those functions g , and, in particular, show the $g(\mathbf{x}) = 0$ contour that separates $[-1.5, 1.0] \times [-.2, 1.3]$ into the regions where a corresponding SVM classifies to classes -1 and 1 .

b) Have a look at the Culp, Johnson, and Michailidis paper referred to in Problem 5 of Section A.28. It provides perspective and help with both the `ada` and `randomForest` packages. Find both `AdaBoost.M1` and `random forest` classifiers appropriate for the Ripley example. For a fine grid of points in $[-1.5, 1.0] \times [-.2, 1.3]$, indicate on a 2-d plot which points get classified to classes -1 and 1 so that you can make visual comparisons to the SVM classifiers referred to in **a**).

2. (6E2-11) In what specific way(s) does the use of kernels and SVM methodology typically lead to identification of a *small* number of important features (basis functions) that are effective in 2-class classification problems?

3. (6HW-13) Consider again the Swiss Bank Note dataset of Problem 4 of Section A.28. Use `caret train()` to choose (via LOOCV) to choose a SVM based on Gaussian kernel and 0-1 loss for these data. Compare its training set and cross-validation error rates to what you found in the Problem 4 of Section A.28 and Problem 2 of Section A.30.

4. (6HW-13) Repeat part **a)** of Problem 1 above on the Seeds data of Problem 5 Section A.28. (Do the plotting on $[-3, 3] \times [-3, 3]$.)

5. (6E2-13) Consider again the toy classification scenario of Problem 6 in Section A.28.

- a)** Is there a linear classifier based directly/only on x with $\overline{\text{err}} = 0$? Explain.
- b)** Is there a support vector machine classifier based on the kernel $\mathcal{K}(x, z) = (1 + xz)^2$ with $\overline{\text{err}} = 0$? Explain.
- c)** Is there a support vector machine classifier based on the kernel $\mathcal{K}(x, z) = \exp(-2(x - z)^2)$ that has $\overline{\text{err}} = 0$? Explain.

6. (5HW-14) Return to the context of Problem 7 Section A.2, Problem 1 Section A.5, and Problem 5 in Section A.23 and the $N = 400$ training set and large test set.

a) Apply linear discriminant analysis to the training set. Identify the regions in $(0, 1)^2$ corresponding to the values of $\hat{y} = 1, 2, 3, 4$. Evaluate the (conditional on the training set) test error rate for LDA based on this training set.

b) Use logistic regression (e.g. as implemented in `glm()` or `glmnet()`) on the training data to find 6 classifiers with linear boundaries for choice between all pairs of classes. Then consider an OVO classifier that classifies \mathbf{x} to the class with the largest sum (of 3) estimated probabilities coming from these logistic regressions. Make a plot showing the regions in $(0, 1)^2$ where this classifier has $\hat{f}(\mathbf{x}) = 1, 2, 3,$ and 4. Use the large test set to evaluate the (conditional on the training set) error rate of this classifier.

c) It seems from the `glmnet()` documentation that using `family="multinomial"` one can fit multivariate versions of logistic regression models. Try this using the training set. Consider the classifier that classifies \mathbf{x} to the class with the largest estimated probability. Make a plot showing the regions in $(0, 1)^2$ where this classifier has $\hat{f}(\mathbf{x}) = 1, 2, 3,$ and 4. Use the large test set to evaluate the (conditional on the training set) error rate of this classifier.

d) Pages 360-361 of K&J indicate that upon converting output y taking values in $\{1, 2, 3, 4\}$ to 4 binary indicator variables, one can use `nnet` with the 4 binary outputs (and the option `linout = FALSE`) to fit a single hidden layer neural network to the training data with predicted output values between 0 and 1 for each output variable. Try several different numbers of hidden nodes and "decay" values to get fitted neural nets. From each of these, define a classifier that classifies \mathbf{x} to the class with the largest predicted response. Use the large test set to evaluate (conditional) error rates of these classifiers and pick the one

with the smallest. Make a plot showing the regions in $(0, 1)^2$ where your best neural net classifier has $\hat{f}(\mathbf{x}) = 1, 2, 3,$ and 4 .

e) Use `svm()` in package `e1071` to fit SVMs to the $y = 1$ and $y = 2$ training data for the

- "linear" kernel,
- "polynomial" kernel (with default order 3),
- "radial basis" kernel (with default gamma, half that gamma value, and twice that gamma value)

Use the `plot()` function to investigate the nature of the 5 classifiers. Put the training data pairs on the plot using different symbols or colors for classes 1 and 2, and also identify the support vectors.

f) Find SVMs (using the kernels indicated in d)) for the $K = 4$ class problem. Again, use the `plot()` function to investigate the nature of the 5 classifiers. Use the large test set to evaluate the (conditional) error rates for these 5 classifiers.

g) Use either the `ada` package or the `adabag` package and fit an AdaBoost.M1 classifier to the $y = 1$ and $y = 2$ training data. Make a plot showing the regions in $(0, 1)^2$ where this classifier has $\hat{f}(\mathbf{x}) = 1$ and 2 . Use the large test set to evaluate the conditional error rate of this classifier. How does this error rate compare to the best possible one for comparing classes 1 and 2 with equal weights on the two? (You should be able to get the latter analytically.)

h) It appears from the Culp, Johnson, and Michailidis paper referred to in Problem 5 of Section A.28 that `ada` implements a OVA version of a K -class AdaBoost classifier in R. Use this and find the corresponding classifier. Make a plot showing the regions in $(0, 1)^2$ where this classifier has $\hat{f}(\mathbf{x}) = 1, 2, 3,$ and 4 . Use the large test set to evaluate the conditional error rate of this classifier.

7. (5E2-14) Consider a 2-class 0-1 loss classification problem with $\{-1, 1\}$ coding of y . For input $\mathbf{x} \in \mathbb{R}^2$ and a parameter $\gamma > 0$, based on a training set of size N consider the classifier

$$\hat{f}(\mathbf{x}) = \begin{cases} 1 & \text{if } \sum_{\substack{i \text{ with} \\ y_i=1}} \exp(-\gamma \|\mathbf{x} - \mathbf{x}_i\|^2) > \sum_{\substack{i \text{ with} \\ y_i=-1}} \exp(-\gamma \|\mathbf{x} - \mathbf{x}_i\|^2) \\ -1 & \text{if } \sum_{\substack{i \text{ with} \\ y_i=1}} \exp(-\gamma \|\mathbf{x} - \mathbf{x}_i\|^2) < \sum_{\substack{i \text{ with} \\ y_i=-1}} \exp(-\gamma \|\mathbf{x} - \mathbf{x}_i\|^2) \end{cases}$$

a) On what basis might one expect that for large N this classifier is approximately optimal?

b) For what "voting function" $g(\mathbf{x})$ is $\hat{f}(\mathbf{x}) = \text{sign}(g(\mathbf{x}))$? Is this $g(\mathbf{x})$ a linear combination of radial basis functions?

c) Why will $\hat{f}(\mathbf{x})$ typically *not* be of the form of a support vector machine based on a Gaussian kernel?

A.34 Section 14 Exercises

1. (6E2-11) The reduced rank classifier of Problem 2 of Section A.29 can be thought of as a "prototype classifier." Give 4 prototypes (real numbers) that can be thought of as defining the classifier.

2. (6HW-13) Return to the context of Problem 5 of Section A.28 and the Seeds dataset. Use the K -means method in the R `stats` package (or some other equivalent method) to find K (7-dimensional) prototype vectors for representing each of the 3 wheat varieties for each of $K = 5, 7, 10$. Then compare training error rates for classifiers that classify to the variety with the nearest prototype for these values of K .

3. (6HW-17) Consider again the Wisconsin breast cancer dataset of Problem 24 of Section A.2. In what follows use standardized versions of the $p = 9$ inputs.

a) For both malignant cases and (separately) benign cases, use K -means clustering and by considering error sums of squares across the inputs, identify small values of K beyond which more clusters are "not essential" in representing the data cases. Make parallel coordinates plots (if you aren't familiar with these, see e.g. <https://datascience.blog.wzb.eu/2016/09/27/parallel-coordinate-plots-for-discrete-and-categorical-data-in-r-a-comparison/>) for the $K_{\text{malignant}}$ and K_{benign} mean vectors produced. (Use the same vertical scales on the two plots so that you can compare them.)

b) Using the $K_{\text{malignant}} + K_{\text{benign}}$ mean vectors identified part a) as prototypes, classify the cases in the dataset according to whether the closest prototype represents a malignant or a benign case. What are the training error rates (malignant, benign, and overall)?

c) Follow the LVQ algorithm as outlined in the exposition for 1000 iterations beginning from the $K_{\text{malignant}} + K_{\text{benign}}$ mean vectors identified part a) as prototypes. Use a series of learning rates $\varepsilon_m = .1(.999)^{m-1}$. Then classify the cases in the dataset according to whether the closest prototype represents a malignant or a benign case. What are the training error rates (malignant, benign, and overall)?

A.35 Section 15.2 Exercises

1. (6HW-11) Let \mathcal{A} be the set of absolutely continuous functions on $[0, 1]$ with square integrable first derivatives (that exist except possibly at a set of measure 0). Equip \mathcal{A} with an inner product

$$\langle h, g \rangle_{\mathcal{A}} = h(0) + g(0) + \int_0^1 h'(x) g'(x) dx$$

a) Show that

$$R(x, z) = 1 + \min(x, z)$$

is a reproducing kernel for this Hilbert space of functions.

b) Using Heckman's development, describe as completely as possible

$$\arg \min_{h \in \mathcal{A}} \left(\sum_{i=1}^N (y_i - h(x_i))^2 + \lambda \int_0^1 (h'(x))^2 dx \right)$$

c) Using Heckman's development, describe as completely as possible

$$\arg \min_{h \in \mathcal{A}} \left(\sum_{i=1}^N \left(y_i - \int_0^{x_i} h(t) dt \right)^2 + \lambda \int_0^1 (h'(x))^2 dx \right)$$

2. (6HW-15) In the context of Problem 1 above, consider the toy dataset below.

y	1.1	1.5	2.4	2.2	1.7	1.3	.3	.1	.1	.5	.1
x	0	.1	.2	.3	.4	.5	.6	.7	.8	.9	1.0

a) For two different values of $\lambda > 0$ find the optimizing function $h \in \mathcal{A}$ for the criterion in part b) of Problem 1.

b) For two different values of $\lambda > 0$ find the optimizing function $h \in \mathcal{A}$ for the criterion in part c) of Problem 1.

A.36 Section 15.3 Exercises

1. (6E2-15) Consider the Gaussian kernel $\mathcal{K}(x, z) = \exp(-(x-z)^2)$ for x and z in $[-2, 4]$ and a corresponding RKHS, \mathcal{A} . Based on the very small (x, y) training set

y	4	4	3	3	2
x	-1	0	1	2	3

we wish to fit a function of the form $\hat{f}(x) = \alpha_0 + \alpha_1 x + h(x)$ for $h \in \mathcal{A}$ under the fitting criterion

$$\sum_{i=1}^5 (y_i - \hat{f}(x_i))^2 + 2 \|h\|_{\mathcal{A}}^2$$

You may use the fact that the least squares line through these data pairs is $\hat{y} = 3.7 - .5x$. Find the optimizing $\hat{f}(x)$.

2. (6HW-17) Return to the baseball home run dataset of Problem 7 of Section A.13 (treating the year index as "x"). Consider the two kernel functions $\mathcal{K}_1(x, z) = \exp(-.5(x-z)^2)$ and $\mathcal{K}_2(x, z) = \exp(-(x-z)^2)$ and the corresponding RKHSs (say \mathcal{A}_1 and \mathcal{A}_2). For $\lambda = 1$ and $\lambda = 10$ find coefficients β_0, β_1 , and β_2 and function $h \in \mathcal{A}$ minimizing

$$\sum_{i=1}^N (y_i - (\beta_0 + \beta_1 x + \beta_2 x^2 + h(x)))^2 + \lambda \|h\|_{\mathcal{A}}^2$$

(There are 4 different optimizations intended here for the two kernels and two values of λ .) Plot the 4 resulting functions

$$\beta_0 + \beta_1 x + \beta_2 x^2 + h(x)$$

on a single set of axes, together with the 145 original (x, y) data points. (If this is computationally infeasible, you may reduce the size of the problem by considering only the "last N " years in the dataset, where N doesn't break your computer.)

A.37 Section 15.4 Exercises

1. (6HW-11) Center the outputs for the dataset of Problem 1 of Section A.18. Then derive sets of predictions \hat{y}_i based on $\mu(x) \equiv 0$ Gaussian process priors for $f(x)$. Plot several of those as functions on the same set of axes (along with centered original data pairs) as follows:

a) Make one plot for cases with $\sigma^2 = 1, \rho(\Delta) = \exp(-c\Delta^2), \tau^2 = 1, 4$, and $c = 1, 4$.

b) Make one plot for cases with $\sigma^2 = 1, \rho(\Delta) = \exp(-c|\Delta|), \tau^2 = 1, 4$, and $c = 1, 4$.

c) Make one plot for cases with $\sigma^2 = .25$, but where otherwise the parameters of **a)** are used.

2. (6HW-11) Consider again the situation of Problem 1 Section A.16. Center the outputs and then derive a set of predictions \hat{y}_i based on a $\mu(x) \equiv 0$ prior for $f(x)$. Use $\sigma^2 = (.02)^2, \rho(x-z) = \exp(-2\|x-z\|^2)$, and $\tau^2 = .25$. How do these compare to the ones you made in Section A.16?

3. (6HW-13) Consider again the situation of Problem 2 Section A.16. Center the outputs and then derive a set of predictions \hat{y}_i based on a $\mu(x) \equiv 0$ prior for $f(x)$. (Use $\rho(x-z) = \exp(-c\|x-z\|^2)$ and what seem to you to be appropriate values of c, σ^2 , and τ^2 .) How do your predictions compare to the ones you made in Section A.16?

4. (6HW-15) Consider again the situation of Problem 17 Section A.2. Center the outputs and then derive a set of predictions \hat{y}_i based on a $\mu(x) \equiv 0$ prior for $f(x)$. Plot several of those as functions on the same set of axes (along with centered original data pairs) as follows:

a) Make one plot for cases with what appear to you to be a sensible choice of σ^2 , for $\rho(\Delta) = \exp(-c\Delta^2), \tau^2 = \sigma^2, 4\sigma^2$, and $c = 1, 4$.

b) Make one plot for cases with $\rho(\Delta) = \exp(-c|\Delta|)$ and the choices of parameters you made in **a)**.

c) Make one plot for cases with σ^2 one fourth of your choice in **a)**, but where otherwise the parameters of **a)** are used.

A.38 Section 17.1 Exercises

1. (6HW-13) There is a small fake dataset below. It purports to be a record of 20 transactions in a drugstore where toothpaste, toothbrushes, and shaving cream are sold. Assume that there are 80 other transaction records that include no purchases of any toothpaste, toothbrush, or shaving cream.

Transaction	Toothpaste			Toothbrush			Shaving Cream		
	Crest	Colgate	AIM	Oral B	Crest	GUM	Barbasol	Old Spice	Gillette
1	X				X				
2				X					
3							X		
4		X					X		
5								X	
6	X				X				
7			X			X		X	
8		X				X			
9			X	X					
10	X						X		
11		X							X
12	X			X					
13			X						
14			X			X	X		
15		X			X				X
16	X				X			X	
17				X					
18			X	X				X	
19		X							X
20				X				X	

- Find $I^{.02}$ (the collection of item sets with support at least .02).
- Find all association rules derivable from rectangles in $I^{.02}$ with confidence at least .5.
- Find the association rule derivable from a rectangle in $I^{.02}$ with the largest lift.

2. (5E2-14) In a toy transaction database there are 5 transactions with items from the set of letters A through G. These are:

Transaction Number	Items Included
1	A,B,D,G
2	B,C,E,G
3	A,C,D,F
4	C,D,E,G
5	A,B,C,G

- Find all item sets of support at least .4.
- For the 3-item set with the largest support, what are the confidence, expected confidence and lift of the associated conjunctive rules?

A.39 Section 17.2 Exercises

1. (6HW-13) Work again with the Seeds data of Problem 5 Section A.28. Begin by again standardizing all $p = 7$ measured variables. JMP will do clustering

for you. (Look under the **Analyze**->**Multivariate** menu.) In particular, it will do both hierarchical and K -means clustering, and even self-organizing mapping as an option for the latter. Consider

- i) several different K -means clusterings (say with $K = 9, 12, 15, 21$),
- ii) several different hierarchical clusterings based on 7-d Euclidean distance (say, again with $K = 9, 12, 15, 21$ final clusters), and
- iii) SOMs for several different grids (say $3 \times 3, 3 \times 5, 4 \times 4$, and 5×5).

Make some comparisons of how these methods break up the 210 data cases into groups. You can save the clusters into the JMP worksheet and use the **GraphBuilder** to quickly make plots. If you "jitter" the cluster numbers and use "variety" for both size and color of plotted points, you can quickly get a sense as to how the groups of data points match up method-to-method and number-of-clusters-to-number-of-clusters (and how the clusters are or are not related to seed variety). Also make some comparisons of the sums squared Euclidean distances to cluster centers.

2. (6HW-13) A $p = 2$ dataset (that has $N = 200$ cases of (x_1, x_2) pairs) with "obvious graphical structure" is provided with these notes. Plot these 200 pairs and see that there are somehow 4 different kinds of "structure" in the dataset. Apply the "graphical spectral features" idea (use $w(d) = \exp(-d^2/c)$) and see if you can "find" the 4 structures in the dataset (by appropriate choice of c and using hierarchical clustering of 200 vectors of 4 or fewer dimensions).

3. (6E2-13) Give a $p = 1$ dataset of size $N = 4$ that shows that the result of ordinary K -means clustering can depend upon the starting cluster centers. (List the 4 data values, consider the 2-cluster problem, and give two different pairs of starting centers that produce different final clusterings. Your starting centers do not need to be data points.)

4. (6E2-13) Below is a toy proximity matrix for $N = 6$ items. Show the steps of agglomerative hierarchical clustering (from 5 to only 2 clusters) using both single and complete linkage. (At every stage, list the clusters as subsets of $\{1, 2, 3, 4, 5, 6\}$. In case of "ties" at any step, pick any of the equivalent possibilities.)

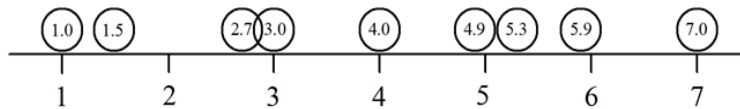
$$\begin{pmatrix} 0 & 1 & 1 & 1.41 & 1.41 & 1.74 \\ 1 & 0 & 1.40 & 1.01 & 1.73 & 1.41 \\ 1 & 1.40 & 0 & 1.72 & 1.01 & 1.41 \\ 1.41 & 1.01 & 1.72 & 0 & 1.40 & 1 \\ 1.41 & 1.73 & 1.01 & 1.40 & 0 & 1 \\ 1.74 & 1.41 & 1.41 & 1 & 1 & 0 \end{pmatrix}$$

5. (5HW-14) Apply model-based clustering to the "USArrests" data in basic R using the `mclust` package and interpret your results.

6. (6HW-17) Consider again the "Glass Identification" dataset of Problem 6 of Section A.2. Use `mclust` (for Gaussian model-based clustering) and `hclust` (for

hierarchical clustering) using average linkage to cluster the 215 glass samples on the basis of the 7 (standardized) inputs used there. For the case of 6 clusters from each method, make a table giving counts of cases in a given cluster from `mclust` and a given cluster from `hclust`. Then compute the "Rand index" for comparing clusterings (look it up on Wikipedia).

7. (5E2-14) Below is a representation of a toy 9-point dataset with $p = 1$. Use agglomerative hierarchical clustering first with single linkage and then with complete linkage to find $K = 3$ clusters in these values. *List for each agglomeration step all groups of more than one value.* (You don't need to list every value in the dataset.)



8. (5HW-20) Consider the problem of clustering points $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \dots, \mathbf{x}_r$ belonging to \mathfrak{R}^p after transforming them to an abstract function space on \mathfrak{R}^p using the mapping $T(\mathbf{x})(\cdot) = \mathcal{K}(\mathbf{x}, \cdot) = \exp(-\gamma \|\mathbf{x} - \cdot\|^2)$, where the function space inner product for points mapped from \mathfrak{R}^p is $\langle T(\mathbf{x}), T(\mathbf{z}) \rangle_{\mathcal{A}} = \mathcal{K}(\mathbf{x}, \mathbf{z})$. Suppose that squared function-space distance is the dissimilarity measure used.

a) Describe agglomerative hierarchical clustering in enough detail that it could be implemented from any formulas and instructions that you supply.

b) Describe K -means clustering in the function space in enough detail that it could be implemented from any formulas and instructions that you supply. (Notice that the concept of arithmetic average makes sense in any linear space, including the abstract feature space.)

A.40 Section 17.3 Exercises

1. (6HW-13) Use appropriate R packages/functions and do multi-dimensional scaling on the 210 cases of the Seeds dataset used in Problem 5 Section A.28, mapping from \mathfrak{R}^7 to \mathfrak{R}^2 using Euclidean distances. Plot the 210 vectors $\mathbf{z}_i \in \mathfrak{R}^2$ using different plotting symbols for the 3 different varieties.

2. (6E2-13) Below is a toy proximity matrix for $N = 4$ items. If one should want to map items to \mathfrak{R}^1 in a way that makes distances between corresponding points in \mathfrak{R}^1 approximately equal to the dissimilarities in the matrix, there is no loss of generality in assuming that the first item is mapped to $z_1 = 0$. Say why there is then no loss of generality to assume that that the second item is mapped to a positive value, i.e. $z_2 > 0$ and provide a suitable function of z_2, z_3 , and z_4 that you would try to optimize in order to accomplish this task.

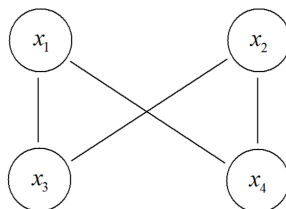
$$\begin{pmatrix} 0 & 1 & 1 & \sqrt{2} \\ 1 & 0 & \sqrt{2} & 1 \\ 1 & \sqrt{2} & 0 & 1 \\ \sqrt{2} & 1 & 1 & 0 \end{pmatrix}$$

3. (6HW-17) The 10 countries in the world with the largest populations are China, India, United States, Indonesia, Brazil, Pakistan, Nigeria, Bangladesh, Russia, and Mexico. You can find the (great circle) distances between their capital cities using this online calculator:

<http://www.chemical-ecology.net/java/capitals.htm> Use multi-dimensional scaling to make a 2-d representation of these cities intended to more or less preserve great circle distances. (The pattern at <http://www.personality-project.org.html> might prove helpful to you.)

A.41 Section 18.2.1 Exercises

1. (6E2-13) Below is a network diagram for a simple restricted Boltzmann machine (with hidden nodes 1 and 2, and visible nodes 3 and 4).. Assume the corresponding probability model for $\mathbf{x} = (x_1, x_2, x_3, x_4)$ has parameters $\theta_{01}, \theta_{02}, \theta_{03}, \theta_{04}, \theta_{13}, \theta_{14}, \theta_{23}$, and θ_{24} and that somehow the network has been "trained" producing $\hat{\theta}_{01} = \hat{\theta}_{02} = 1, \hat{\theta}_{03} = \hat{\theta}_{04} = -1, \hat{\theta}_{13} = \hat{\theta}_{14} = 1$, and $\hat{\theta}_{23} = \hat{\theta}_{24} = -1$.



- a) Find (for the fitted model) the ratio $P[\mathbf{x} = (1, 0, 1, 0)] / P[\mathbf{x} = (0, 0, 0, 0)]$.
- b) Find (for the fitted model) the conditional distribution of (x_1, x_2) given that $(x_3, x_4) = (0, 0)$. (You will need to produce 4 conditional probabilities.)

A.42 "General/Comprehensive" Exercises

1. (5HW-20) Consider the White Wines Dataset⁵⁶ from the UCI Machine Learning Data Repository

<http://archive.ics.uci.edu/ml/datasets/Wine+Quality>.

Consider SEL prediction of what can be learned about wine "quality" from the 11 input variables. There are roughly 5000 cases in this dataset, and it is about at the (size) limit of what is conveniently handled using R and an ordinary laptop. (Other faster software like Python or MatLab and/or implementation on a server or cluster may be required for bigger datasets with many machine learning applications.)

⁵⁶The White Wines Dataset is not absolutely ideal as an example in that the response variable can take only integer values 1 through 10 and is probably not really an interval-level variable in the first place (being more ordinal in nature). For purposes of exercise we will ignore these matters, and treat the quality rating as a measured numerical response and consider prediction under SEL.

a) Find sets of best (according to LOOCV) predictions for the quality ratings for

- k -nn prediction
- elastic net prediction
- PCR prediction
- PLS prediction
- MARS prediction (implemented in `earth`)
- regression tree prediction
- random forest prediction
- boosted trees prediction
- Cubist prediction

Say what parameters you settled on for each method.

b) Make a scatterplot matrix for all 9 sets of prediction in a) plus the y values and OLS predictions. Compute a correlation matrix for these 11 sets of values and display this rounded to 2 decimal places.

c) Consider the problem of combining the 9 "basic" prediction methodologies employed in a) via stacking/generalized stacking/meta-prediction/super-learning. There is nothing that says that the "good" sets of parameters you developed for "individual" use of the prediction methods are in any sense "good" choices if ultimately one is going to use the methods as elements of an "ensemble." But for purposes of exercise here, we are not going to "redo" them, but will take them as chosen. (We will here consider combining these through the use of first OLS MLR and then through the use of a random forest made with "default" parameters.)

Randomly break the White Wines dataset into 10 folds of sizes as nearly equal as possible. For each fold and its remainder fit a predictor using the remainder as a training set via each of the methods in a) (and the parameters of the methods previously identified) and use it to make predictions for cases in the fold. Then

1. Use the remainder as a training set and the values of the 9 predictors (on the remainder) as "features" in a MLR model (including intercept). Use OLS to fit this to the outputs for the cases in the remainder.
2. Use the remainder as a training set and the values of the 9 predictors (on the remainder) as "features" and fit a default random forest to the outputs for the cases in remainder.
3. Apply the coefficients from 1. to the 9 predictions to make an ensemble prediction for each case in the fold.

4. Apply the random forest fit in 2. to the 9 predictions to make an ensemble prediction for each case in the fold.
5. For both the predictions in 3. and 4. add the squared differences between outputs and predicted outputs across the fold.
6. Total the results of 5. across the 10 folds, divide by N , and take a square root to get a "RMSPE" for the basic methods and parameters combined through OLS and through a random forest.

Do the values "RMSPE" in 6. improve on what you have for the best of the CVRMSPEs for the individual methods?

d) Discuss how you would get an "honest" CV assessment of likely performance of the strategy of first fitting predictors using methods in **a)** obtaining parameters from `caret train()` and then combining them via OLS MLR or default random forest. Explain why the "RMSPE" values from **c)** are probably too optimistic to serve the purpose here.

2. (5HW-20) Consider again the Glass Identification dataset and 2-class classification problem of Problem 6 of Section A.2.

a) Use LOOCV to identify a good number of neighbors to use for k -nn classification (based on 0-1 loss) for the 2-class classification problem.

b) Use LOOCV to identify a good classification tree for 0-1 loss in the 2-class classification problem.

c) Use the OOB error and optimize a classification random forest over choice of both m and n_{\min} for 0-1 loss in the 2-class classification problem.

d) Use LOOCV to identify a good (single layer feed-forward) neural network for classification (optimize over both number of hidden nodes and weight decay) based on 0-1 loss for the 2-class classification problem.

e) Use LOOCV to identify a good elastic net penalized logistic regression for 0-1 loss 2-class classification.

f) Use LOOCV to identify a good support vector classifier (based on 0-1 loss) for the 2-class classification problem. (That is, find a good SVM with "linear kernel.")

g) Use as much LOOCV grid-searching as you can afford (time-wise) to identify a good support vector machine with "Gaussian kernel" (based on 0-1 loss) for the 2-class classification problem.

h) Use as much LOOCV grid-searching as you can afford (time-wise) to identify a good number of iterations for an AdaBoost.M1 classifier based on 0-1 loss for the 2-class classification problem.

i) Use as much LOOCV grid-searching as you can afford (time-wise) to identify a good tree-boosting classifier using `XGBoost` for 0-1 loss 2-class classification.

j) Compare the classifiers in parts **a)** through **i)** on the basis of

- training error rates (for 0-1 loss)

- the AUC criterion⁵⁷, and
- cross-validation (and OOB) 0-1 loss error rates.

3. (5HW-20). Consider again the White Wines dataset of Problem 1 above, but now the problem of predicting the class variable

$$y^* = I[y \geq 7]$$

Call a wine with rating 7 or better a "good" wine and this becomes a problem of classification of wines into "not good" and "good" ones.

a) Carry out the steps **a)** through **i)** in Problem 2 above (there referring to the Glass-Identification problem) for this wine classification problem.

Consider the problem of combining basic classification methodologies via stacking/generalized stacking/meta-prediction/super-learning in the White Wines classification problem immediately above.

b) Use the outputs of your classifiers developed in **a)** and the original input variables (the 11 quality measures giving 9 + 11 "features" in total) as inputs to a default random forest. (Where they are available, use estimated conditional probabilities for class 2 rather than the classification values assigned to the training cases by the classifiers.) What "training error rate" is produced for 0-1 loss? There is a nominal random forest "OOB error" rate associated with your final "super-learner." Why should you NOT trust either of these numbers as being indicative of the likely performance of the "tune 9 classifiers and plug their outputs into a default random forest" prediction methodology?

c) Say very clearly and carefully how (given plenty of computing power) you would compute an honest assessment of the likely performance of the "super-learner" described above.

⁵⁷You may, for example, use the `pROC` package to (plot the "ROC curve" and) compute this.