Statistical Machine Learning-10

More Generalities About Classification, Likelihood Ratios, Voting Functions, and Near Neighbors

Stephen Vardeman

Analytics Iowa LLC

January 2018

Optimal Classification/Pattern Recognition

Minimizing Error Rate

In classification problems where y takes values in $\{0, 1, \ldots, K-1\}$ or $\{1, \ldots, K\}$ (for the $K = 2$ case, the possibility $\{-1, 1\}$ is also quite useful because it simplifies some formulas) a standard loss is 0-1 loss,

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

Expected 0-1 loss is $\mathrm{E}I[\hat{y} \neq y] = PI[\hat{y} \neq y]$, the **misclassification rate**, and a theoretically **optimal classifier** is $f(\mathbf{x})$ a k minimizing

$$\sum_{v \neq k} P[y = v|\mathbf{x}]$$

or is equivalently $f(\mathbf{x})$ a k *maximizing*

$$P[y = k|\mathbf{x}]$$

(An optimal $f(\mathbf{x})$ is a possible value for y with the largest conditional probability given the observed value $\mathbf{x}$.)

Some Generalities
Form of the Conditional Probability of Class k

The optimal classifier is not realizable in practice. But before considering what can be done in practice, it will be useful to make some observations about the nature of the classification/"pattern recognition" problem and its optimal solution that help place later specific methods into perspective.

If $g_k(\mathbf{x})$ is the density (or probability mass function) for $\mathbf{x} \in \Re^p$ conditional on $y = k$ and

$$\pi_k = P[y = k]$$

then as $\sum_{l=1}^{K} \pi_l g_l(\mathbf{x})$ is the marginal pdf of $\mathbf{x}$, and

$$P[y = k|\mathbf{x}] = \frac{\pi_k g_k(\mathbf{x})}{\sum_{l=1}^{K} \pi_l g_l(\mathbf{x})}$$

Some Generalities

Equivalent Form of an Optimal Classifier

Since $\sum_{l=1}^{K} \pi_l g_l(\mathbf{x})$ doesn't depend upon k, the forms of an optimal classifier and the conditional distribution of y given $\mathbf{x}$ imply that an equivalent form for an optimal classifier is $f(\mathbf{x})$ a k maximizing

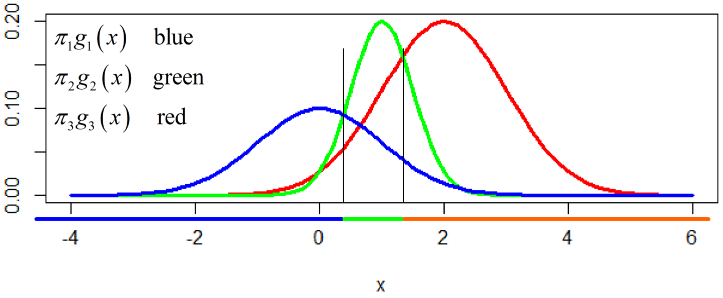
$$\pi_k g_k(\mathbf{x})$$

From one point of view, the prescription "$\mathbf{x}|y$ has density g_y and $y \sim \pi$" has the form of an ordinary Bayes statistical model for $\mathbf{x}$ with "parameter" y and prior π. In this sense, an optimal classifier is a "**Bayes**" **classifier**.

Some Generalities

An Artificial p=1 Example

Consider an artificial $p = 1$ example where $K = 3$, $\pi_1 = .25$, $\pi_2 = .25$, and $\pi_3 = .5$ and the three class-conditional distributions are g_1 normal with $\mu_1 = 0$ and $\sigma_1 = 1$, g_2 normal with $\mu_2 = 1$ and $\sigma_2 = .5$, and g_3 normal with $\mu_3 = 2$ and $\sigma_3 = 1$. The plot below shows the 3 functions $\pi_k g_k(x)$ and indicates the optimal classifier in terms of three sets of x values corresponding to different optimal classifications.



Some Generalities

More on the Example

To help understanding of the form of $P[y=k|\mathbf{x}]$ in classification models, consider again the graphic. For a fixed x, reading off the values of $\pi_1 g_1(x)$ (blue), $\pi_2 g_2(x)$ (green), and $\pi_3 g_3(x)$ (red) from the curves, the values $P[y=1|x]$, $P[y=2|x]$, and $P[y=3|x]$ are *in proportion to those three values*. For example, since

$$.25g_1(1)=.0605, .25g_2(1)=.1995 \text{ and } .5g_3(1)=.1210$$

and $.0605+.1995+.1210=.3810$, the conditional probabilities for $y=1,2,3$ given $x=1$ are respectively

$$\frac{.0605}{.3810}\approx .159,\ \frac{.1995}{.3810}\approx .523 \text{ and } \frac{.1210}{.3810}\approx .318$$

Some Generalities

2-Class Problems and the Likelihood Ratio

Consider the $K = 2$-class model with the $\{0, 1\}$ coding. An optimal classifier can be written as

$$f(\mathbf{x}) = I\left[\pi_1 g_1(\mathbf{x}) > \pi_0 g_0(\mathbf{x})\right] = I\left[\frac{g_1(\mathbf{x})}{g_0(\mathbf{x})} > \frac{\pi_0}{\pi_1}\right]$$

(Again, I [statement] is 1 if "statement" is true and 0 if it is not.) The statistic $g_1(\mathbf{x}) / g_0(\mathbf{x})$ is called the likelihood ratio statistic and is of fundamental use in the classification model. Of course

$$P[y = 1|\mathbf{x}] = \frac{\pi_1 g_1(\mathbf{x})}{\pi_0 g_0(\mathbf{x}) + \pi_1 g_1(\mathbf{x})} = \frac{\pi_1 \dfrac{g_1(\mathbf{x})}{g_0(\mathbf{x})}}{\pi_0 + \pi_1 \dfrac{g_1(\mathbf{x})}{g_0(\mathbf{x})}}$$

and the likelihood ratio and $P[y = 1|\mathbf{x}]$ are increasing functions of each other.

Some Generalities

More on 2-Class Models and the Likelihood Ratio

The likelihood ratio statistic is important in 0-1 loss versions of the $K = 2$-class classification problem in that it provides an optimal function whereby one decides whether or not to classify to class 1, large values pointing to the classification 1. And it is fundamental from other perspectives as well. In a 2-class model where the so-called "area under the curve" criterion is used to judge a choice of ordering function $\mathcal{O}$ used to order the values of $\mathbf{x}$ (and thus training cases) in terms of their indication that a corresponding y is 1, the likelihood ratio statistic (or anything equivalent to it like $P\left[y = 1|\mathbf{x}\right]$) is optimal.

Some Generalities

Toy Example of the Likelihood Ratio

To make clear what is meant by the ratio $g_1(\mathbf{x})/g_0(\mathbf{x})$, below are $K = 2$ hypothetical probability mass functions for $(p = 2)$ observations $\mathbf{x}$. The likelihood ratio gives the proper ordering of the 9 possible values of $\mathbf{x}$ for classification purposes.

$g_1(\mathbf{x})$

$x_1 \backslash x_2$	1	2	3
3	.1	.2	.3
2	.1	0	.1
1	.05	.1	.05

$g_0(\mathbf{x})$

$x_1 \backslash x_2$	1	2	3
3	.15	.2	.1
2	.01	.14	.05
1	.2	0	.15

$g_1(\mathbf{x})/g_0(\mathbf{x})$

$x_1 \backslash x_2$	1	2	3
3	2/3	1	3
2	10	0	2
1	1/4	∞	1/3

From least indicative of class 1 to most indicative, these are $(2,2)$, $(1,1)$, $(1,3)$, $(3,1)$, $(3,2)$, $(2,3)$, $(3,3)$, $(2,1)$, and $(1,2)$.

Voting Functions and 2-Class Classification

"Voting Functions"

Empirical search for a good classifier in 2-class problems is essentially a search for a good approximation to the likelihood ratio function $g_1(\mathbf{x})/g_0(\mathbf{x})$. This raises the possibility of focusing on the building of a good "voting function" $v(\mathbf{x})$ underlying a classifier.

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

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

(except for the possibility that $v(\mathbf{x}) = 0$, that typically has 0 probability for both classes). An optimal voting function for 0-1 loss is the shifted likelihood ratio

$$v^{\text{opt}}(\mathbf{x}) = \frac{g_1(\mathbf{x})}{g_{-1}(\mathbf{x})} - \frac{\pi_{-1}}{\pi_1} \qquad (1)$$

Voting Functions and 2-Class Classification

"Voting Functions" and 0-1- Loss Error Rate

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

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

(a loss is incurred when y and $v(\mathbf{x})$ have opposite signs). Then, the 0-1 loss error rate has the useful representation

$$\text{E}I[y \cdot v(\mathbf{x}) < 0] \tag{2}$$

A function $v(\mathbf{x})$ optimizing the average value (2) is the shifted likelihood ratio defined in (1). But the indicator function $I[u < 0]$ involved in (2) is discontinuous, 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.

Voting Functions and 2-Class Classification

"Voting Functions," Continuous "Losses," and Bounds on 0-1 Loss Error Rate

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

$$\mathrm{E}I[y \cdot v(\mathbf{x})<0] \leq \mathrm{E}h(y \cdot v(\mathbf{x})) \tag{3}$$

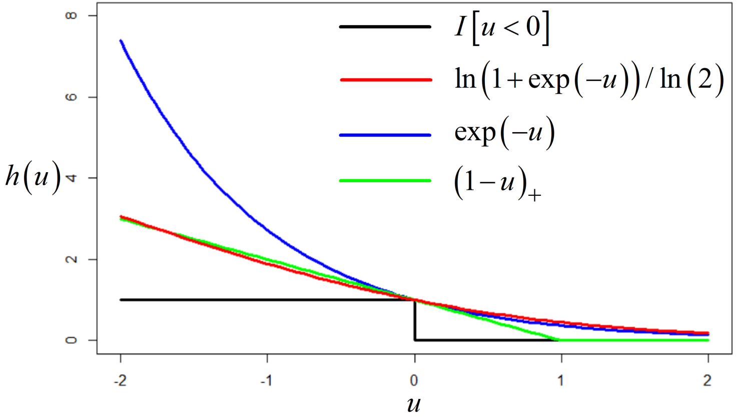
So with voting function $v(\mathbf{x})$, the right hand side of (3) is an upper bound for the 0-1 loss error rate of the corresponding classifier. An approximate (data-based) minimizer (over choices $v(\mathbf{x})$) of rhs (3) can be expected to control 0-1 loss error rate. Several continuous choices of "function-loss" $h(u)$ motivate popular methods of classifier development. These include

1. $h(u)=\ln(1+\exp(-u))/\ln(2)$ associated with use of logistic regression-based estimated conditional class probabilities to make voting functions,
2. $h(u)=\exp(-u)$ associated with the "AdaBoost" algorithm, and
3. $h(u)=(1-u)_{+}$ associated with "support vector machines."

Voting Functions and 2-Class Classification

"Voting Functions," Continuous "Losses," and Bounds on 0-1 Loss Error Rate

For sake of concreteness, below is a plot of $I[u < 0]$ and the three functions $h(u)$ dominating it discussed on the previous slide.



Voting Functions and 2-Class Classification

Optimizing Functions for Standard Expected Continuous Losses

One reason why this line of argument proves effective is that not only does bound (3) hold, but minimizers of $\mathrm{E}h\left(y\cdot v\left(\mathbf{x}\right)\right)$ over choice of function $v\left(\mathbf{x}\right)$ for standard choices of function-loss h with $I\left[u<0\right]\leq h\left(u\right)$ turn out to be directly related to the likelihood ratio. That is, case 1. on the previous slide has optimizing function

$$v^{*}\left(\mathbf{x}\right)=\ln\left(\frac{P\left[y=1|\mathbf{x}\right]}{P\left[y=-1|\mathbf{x}\right]}\right)$$

and case 2. has an optimizer that is 1/2 of this. Both are monotone transformations of the likelihood ratio and when used as a voting function produce a (0-1 loss) optimal classifier. In case 3. from the previous slide, an optimizing function is

$$v^{**}\left(\mathbf{x}\right)=\mathrm{sign}\left(P\left[y=1|\mathbf{x}\right]-P\left[y=-1|\mathbf{x}\right]\right)$$

the optimal classifier itself. **So empirical search for optimizers of (an empirical version of)** $\mathrm{E}h\left(y\cdot v\left(\mathbf{x}\right)\right)$ **can produce good classifiers**.

Near Neighbors and Classification

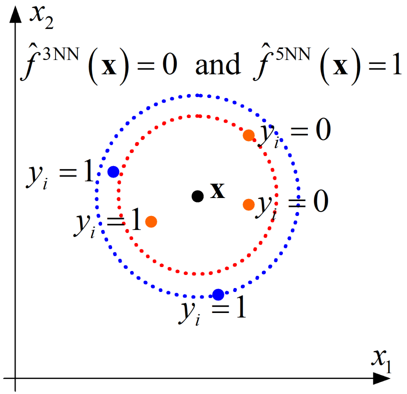
The Form of an Optimal Classifier and Near Neighbors

Consider the part of a very large sample that has $\mathbf{x} \approx \mathbf{x}_0$. The fraction of that part that has $y = k$ will approximate $P\left[y = k \middle| \mathbf{x} = \mathbf{x}_0\right]$. Since an optimal classifier chooses a k maximizing this conditional probability, choosing a k that is most heavily represented among the part of a sample with $\mathbf{x} \approx \mathbf{x}_0$ can approximate an optimal classification at $\mathbf{x}_0$.

So, we've seen that for large training samples (and small p) in SEL prediction problems, the average y for near neighbors of $\mathbf{x}$ provides an approximation to the optimal predictor at $\mathbf{x}$. Now, in classification problems, we see that it is equally true that for **large training samples the most frequently represented class for near neighbors of $\mathbf{x}$ provides an approximation to an optimal classification at $\mathbf{x}$.**

Nearest Neighbor Classification
A Toy Example

Below is a cartoon illustrating 3- and 5-NN classification at **x** (in a 2-class problem with the $\{0, 1\}$ coding).



x_2
$\hat{f}^{3\text{NN}}(\mathbf{x}) = 0$ and $\hat{f}^{5\text{NN}}(\mathbf{x}) = 1$
$y_i = 0$
$y_i = 1$
$\mathbf{x}$
$y_i = 0$
$y_i = 1$
$y_i = 1$
x_1

Nearest Neighbor Classification

An Example from UCI Data Repository

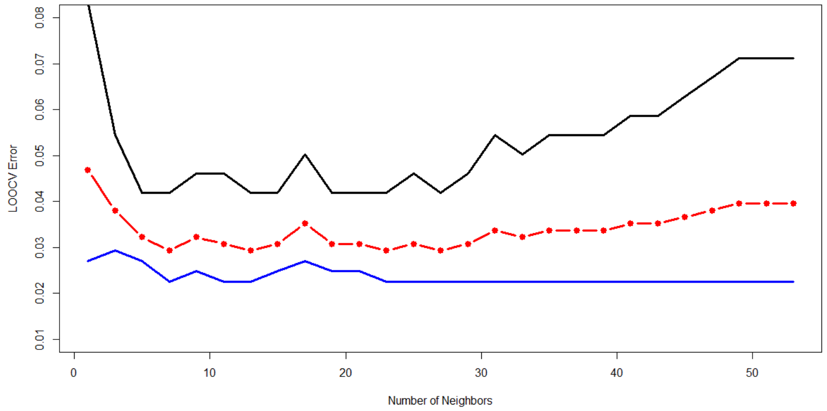
As an example, we'll use a version of the famous "Wisconsin Breast Cancer Study" data set, available at https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/ . This is a dataset with $N = 699 - 16 = 683$ complete cases (16 have missing entries), each one describing $p = 9$ characteristics of a biopsied tumor that has been classified as either benign (444 cases) or malignant (239 cases). The input variables (originally on 1-10 scales) are:

x_1	–Clump Thickness	x_6	–Bare Nuclei
x_2	–Cell Size Uniformity	x_7	–Bland Chromatin
x_3	–Cell Shape Uniformity	x_8	–Normal Nucleoli
x_4	–Marginal Adhesion	x_9	–Mitoses
x_5	–Single Epithelial Cell Size		

Nearest Neighbor Classification

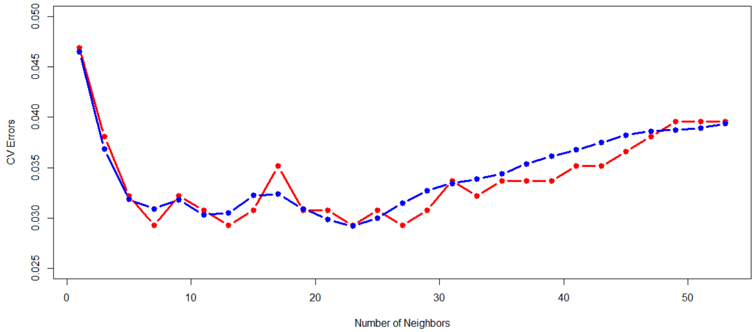
Wisconsin Breast Cancer

LOOCV error rates for kNN classifiers for the Wisconsin Breast Cancer data (overall in red, malignant in black, benign in blue) are below.



Nearest Neighbor Classification
Wisconsin Breast Cancer

A more careful **cross-validation** exercise done (with the `tune()` routine in the `caret` package and 100 repeats of 10-fold cross-validation) *restandardizing after removing each fold* produces essentially the same conclusions about k in this problem. This is evident in the plot below of both the earlier LOOCV error (in red) and the more carefully made CV error (in blue).



Statistical Machine Learning-11

Density Estimates and Approximately Optimal Classifiers

Stephen Vardeman

Analytics Iowa LLC

January 2018

Density Estimates
Motivation

The problem of describing structure for $\mathbf{x} \in \Re^p$ might be phrased in terms of estimating a pdf for the variable. So the problem:

based on $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ iid with (unknown) pdf f, estimate f

is of independent interest. But of even more present importance is the fact that an optimal 0-1 loss classifier is for $\mathbf{x} \in \Re^p$ a k maximizing

$$\pi_k g_k(\mathbf{x})$$

and if one can estimate each $g_k(\cdot)$ based on the part of a training sample with $y = k$ (and approximates each π_k with the fraction of the training sample with $y = k$) an approximately optimal classifier can potentially be made.

Density Estimation in 1-D
Parzen Kernel Density Estimates

Temporarily suppose that $p = 1$. For $\phi(\cdot)$ the standard normal pdf (other choices of basic "kernel" are possible, but this is most common) and a "bandwidth" $\lambda > 0$

$$\frac{1}{\lambda}\phi\left(\frac{\cdot - \theta}{\lambda}\right)$$

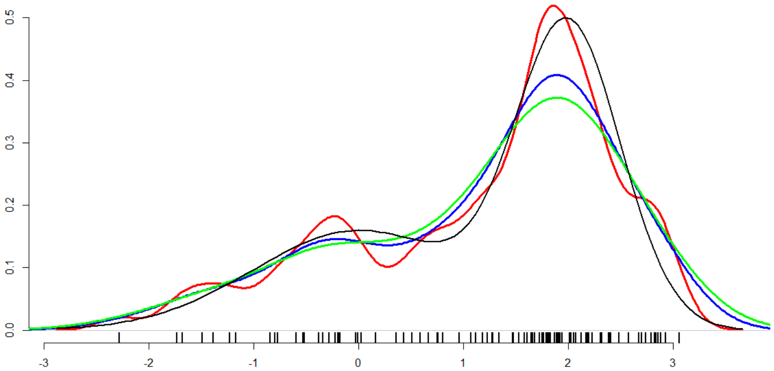
is the normal density for mean θ and standard deviation λ. The Parzen (kernel) estimate of a density at x, $f(x)$, is then

$$\hat{f}_\lambda(x) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{\lambda}\phi\left(\frac{x - x_i}{\lambda}\right)$$

an average of values of normal densities centered at the x_i in a training set.

Density Estimation in 1-D
Example

Below are plots of a pdf, f (in black), a sample of size $N = 100$ from the distribution and (normal kernel) density estimates made with bandwidths $\lambda = .2$ (red),.4 (blue), and .5 (green).



Density Estimation
Normal Kernels

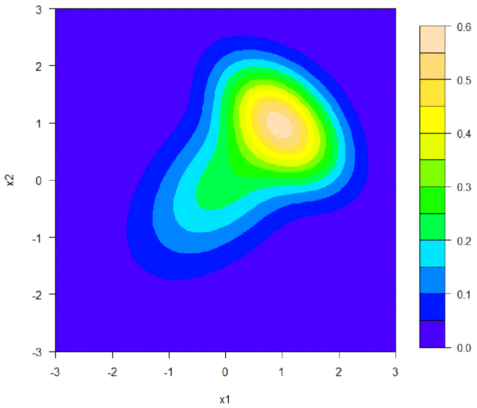
A way to think about the density estimate that results from using a normal kernel is as representing the distribution of "a random choice from the training set perturbed by a mean 0 normal error with standard deviation equal to the bandwidth." If the bandwidth is extremely small, the density estimate 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.

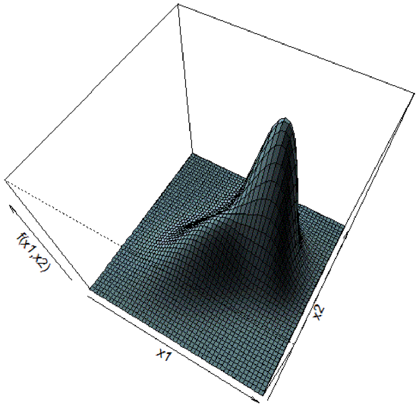
A natural generalization of this to p dimensions is to let $\phi(\cdot)$ be a (mean $\mathbf{0}$) MVN$_p$ density. One should expect that unless N is huge, this methodology will be reliable only for fairly small p (say 3 at most) as a means of estimating a general p-dimensional pdf.

Density Estimation in 2-D

An Example of a 2-D Density

Below are two representations of a particular 2-D density (a mixture of two bivariate Normals).

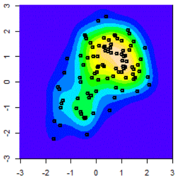
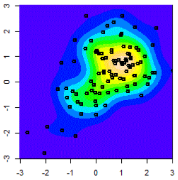
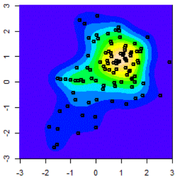
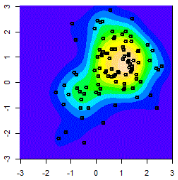
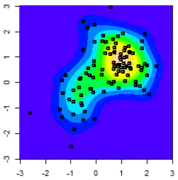
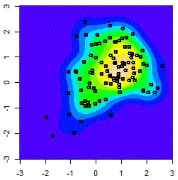




Density Estimation in 2-D

Examples of 2-D Density Estimates

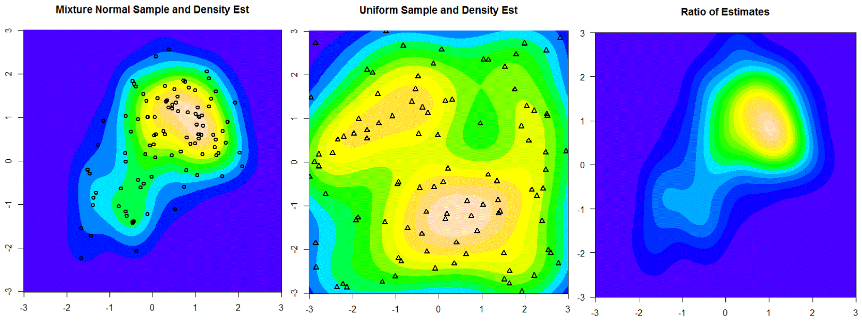
For Illustration, below are 6 samples of $N = 100$ observations from the mixture density pictured on the previous slide and corresponding bivariate density estimates made using the `kde2d` function in the `MASS` package (and its default choice of "bandwidth" covariance matrix).

Density Estimates and Classifiers

An Example of a Ratio of 2-D Density Estimates

The possibility of using direct estimates $\hat{\pi}_k \hat{g}_k(\mathbf{x})$ to make approximately optimal classifiers basically depends upon how well likelihood ratios can be estimated. The graphic below shows for samples of $N = 100$ from the bivariate density on slide 6 and from a uniform density on $[-3,3]^2$ a **ratio of density estimates**.



Density Estimates and Classifiers

Large p and "Naive Bayes" Classifiers

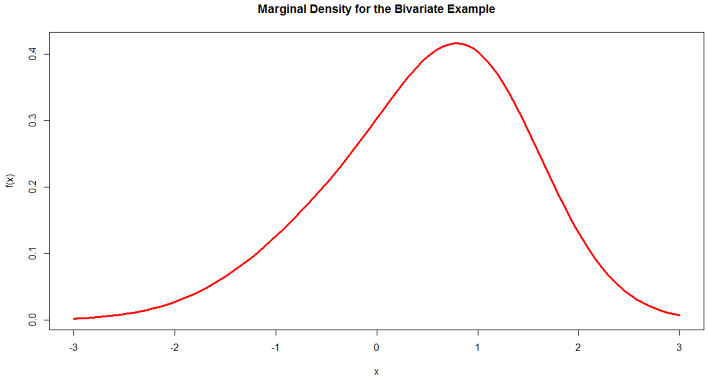
The $p = 2$ example used here looks reasonably hopeful, as the third graph on the previous slide is some approximation of the original density portrayed on slide 6 (which is proportional to its ratio to a uniform/constant density). But the normal mixture and uniform densities are very simple and the curse of dimensionality makes density estimation for even moderate p (let alone estimation of ratios) highly problematic. So **direct approximation of optimal classifiers via density estimates is rarely successful for p at all large**.

One related idea that has proven to be of some use is that of estimating only low-dimensional (small p) marginals of the $g_k(\mathbf{x})$s (for which density estimation is feasible) and making a product of them to substitute for an estimate of the joint density (effectively acting like the input $\mathbf{x}$ can be modeled as having independent pieces) in a classifier. This has been called a **"naive Bayes" classification** method.

Products of Marginals

An Example

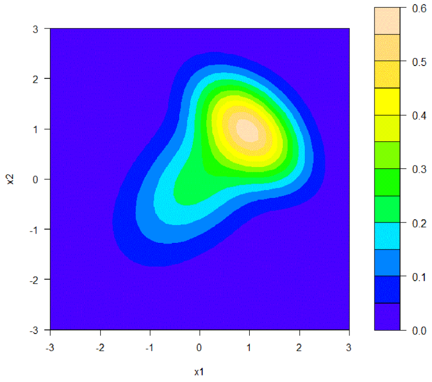
It is easy to see that the naive Bayes idea can fail to be useful even for small p. The density below is the marginal density for both coordinates of $\mathbf{x}$ (both x_1 and x_2) in the bivariate example we have been using. The next panel contrasts the original bivariate density to a density of independence with this one as marginals.

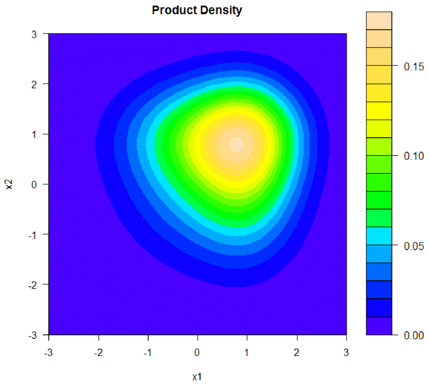


Products of Marginals (and "Naive Bayes")

An Example

The original density is clearly quite different from one of independence with the same marginals.





Statistical Machine Learning-12

Linear Methods of Classification Part 1: Linear Discriminant Analysis and Logistic Regression

Stephen Vardeman

Analytics Iowa LLC

January 2018

"Linear" Classification Rules

Generalities

Continue to suppose y takes values in $\{1, 2, \ldots, K\}$. We consider methods of producing K-class classification rules $\widehat{f}(\mathbf{x})$ (and mostly ones) that have sets $\left\{\mathbf{x} \in \Re^p | \widehat{f}(\mathbf{x}) = k\right\}$ with boundaries that are defined (at least piece-wise) by linear equalities

$$\sum_{j=1}^{p} \beta_j x_j = c \qquad (1)$$

(i.e. $\mathbf{x}'\boldsymbol{\beta} = c$). We consider probability models under which optimal classifiers are "linear," and corresponding methods have classical "statistical" origins and history.

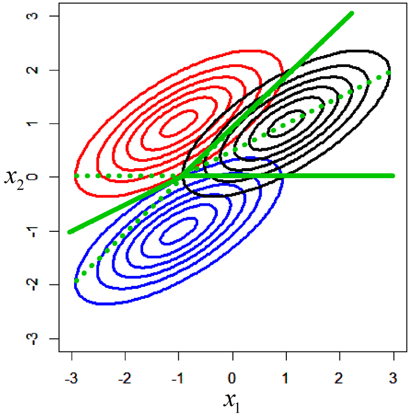
Linear Discriminant Analysis

MVN Class-Conditional Modeling With a Common Covariance Matrix

Suppose that the distribution for $(\mathbf{x}, y)$ has $P[y = k] = \pi_k$ and the conditional distribution of $\mathbf{x}$ on $\Re^p$ given that $y = k$ is $\text{MVN}_p(\boldsymbol{\mu}_k, \boldsymbol{\Sigma})$, i.e. the class-conditional distributions of $\mathbf{x}|y$ are multivariate normal with a common covariance matrix. It then follows that boundaries between regions in $\Re^p$ where $f(\mathbf{x}) = k$ and $f(\mathbf{x}) = l$ (optimal classifications are respectively k and l) are parts of $(p-1)$-dimensional hyperplanes in $\Re^p$ (the exact form of which of course depend upon the K mean vectors $\boldsymbol{\mu}_k$, the values π_k, and the covariance matrix $\boldsymbol{\Sigma}$).

Linear Discriminant Analysis
An Example

Below are contour plots for $K = 3$ bivariate normal densities with a common covariance matrix and the linear classification boundaries corresponding to equal class probabilities $\pi_1 = \pi_2 = \pi_3$.



Practical LDA

Estimating Means and a Common Covariance Matrix

In order to use LDA formulas, one must estimate the vectors $\boldsymbol{\mu}_k$ and the common covariance matrix $\boldsymbol{\Sigma}$ from training data. The sample mean of $\mathbf{x}$ vectors for cases with $y = k$ is the obvious estimate of $\boldsymbol{\mu}_k$ and a pooled sample covariance matrix (made by appropriately combining sample variances and correlations computed within the K parts of the training sample) can be used to estimate $\boldsymbol{\Sigma}$.

Where it is plausible to think of the parts of a training sample corresponding to different classes as producing roughly "football-shaped clouds" of input vectors $\mathbf{x}$ *of similar shapes and orientations* in $\Re^p$, LDA makes good sense. It provides hyperplanes approximately optimally separating those clouds.

LDA Complexity

Dimension Reduction: Forward Selection of Input Coordinates

There are many (roughly $p\left(K+(p+1)/2\right)$) means, variances and correlations to estimate in order to use linear discriminant analysis. Where p is at all big, a training sample may not be big enough to adequately support estimation of this many parameters and LDA based on p-dimensional $\mathbf{x}$ vectors is use of a classifier that is "too complex." There are at least two common strategies for **reducing** dimension or **complexity**.

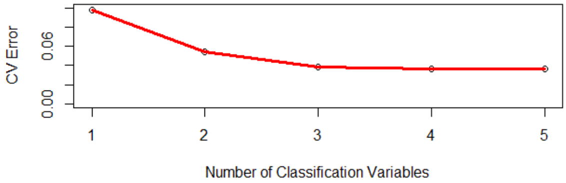
The first method is forward selection of variables x_j for discrimination. This picks the best single coordinate of $\mathbf{x}$ for linear discrimination, then next the coordinate of $\mathbf{x}$ that provides the greatest reduction in training error, etc. **Cross-validation** can be used to choose a good value for the ultimate dimension ($\leq p$) of the input vector used.

LDA Example

Wisconsin Breast Cancer Example

The `lda()` function in the `MASS` package in `R` can be used to do LDA on the Wisconsin breast cancer data. Straightforward use of all $p = 9$ predictors (and thus roughly $9(2+5) = 63$ estimated parameters) in LDA produces LOOCV error rates of $19/239 = 7.59\%$ for malignant cases, $8/444 = 1.8\%$ for benign cases, and 4.39% overall.

On the other hand, 10-fold cross-validation using the `stepclass()` function from the `klaR` package produces the values plotted below showing that p no more than 3 would be a better choice than $p = 9$.



(Variables 6, then 2, then 1 are identified via forward selection.)

LDA Complexity

Dimension Reduction: Use of a Few Canonical Variables

A second method of dimension reduction for LDA is known as "**reduced rank**" **LDA** or **"canonical variables" LDA**. This involves finding a coordinate system that describes variation among the K estimated class means. The first coordinate axis points in the direction of the largest variation in these means, the second (if $K > 2$) axis (perpendicular to the first) points in the direction of the largest part of the remaining variation in the means, and so on to produce $\min(p, K-1)$ axes. Coordinates of data points in this new coordinate system are the variables used in LDA. **The number of such coordinates employed is a complexity measure that can be chosen by cross-validation.**

Use of Features in LDA

Basis Functions/Transforms

The form (1) 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 $(h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_M(\mathbf{x}))$ in the development of LDA.

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 $\mathbf{x}$. Or, data-dependent basis functions that are $K(\cdot, \mathbf{x}_i)$ for some kernel function K produces LDA that depends upon the value of some "basis function network." The possibilities opened here are myriad and "the devil is in the details."

Logistic Regression

Form for Log Conditional Probability Ratio

The MVN distribution result that leads to LDA is that in the common covariance MVN_p model, the ratios

$$\ln\left(\frac{P\left[y=k|\mathbf{x}\right]}{P\left[y=l|\mathbf{x}\right]}\right)$$

are linear in $\mathbf{x}$. An alternative to beginning with the model assumptions of LDA is to simply assume that for all $k<K$

$$\ln\left(\frac{P\left[y=k|\mathbf{x}\right]}{P\left[y=K|\mathbf{x}\right]}\right)=\beta_{k0}+\sum_{j=1}^{p}\beta_{kj}x_j \qquad (2)$$

Here there are $K-1$ constants β_{k0} and $K-1$ (p-dimensional) vectors $\boldsymbol{\beta}_k=(\beta_{k1},\beta_{k2},\ldots,\beta_{kp})$ to be specified (not necessarily tied to class probabilities or mean vectors or a common within-class covariance matrix for $\mathbf{x}$).

Logistic Regression

Conditionals, not Joint

The set of relationships (2) do not fully specify a joint distribution for $(\mathbf{x}, y)$. Rather, they only specify the nature of the conditional distributions of $y|\mathbf{x}$.

The situation is exactly analogous to that in ordinary simple linear regression. A bivariate normal distribution for (x, y) has 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).

Logistic Regression

Implied Conditional Probabilities and Optimal Classifier

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 (2) produces the forms

$$p_k(\mathbf{x}, \boldsymbol{\theta}) = P[y = k|\mathbf{x}] = \frac{\exp\left(\beta_{k0} + \sum_{j=1}^{p} \beta_{kj} x_j\right)}{1 + \sum_{k=1}^{K-1} \exp\left(\beta_{k0} + \sum_{j=1}^{p} \beta_{kj} x_j\right)}$$

for $k < K$, and

$$p_K(\mathbf{x}, \boldsymbol{\theta}) = P[y = K|\mathbf{x}] = \frac{1}{1 + \sum_{k=1}^{K-1} \exp\left(\beta_{k0} + \sum_{j=1}^{p} \beta_{kj} x_j\right)}$$

and an optimal (under 0-1 loss) classifier $f(\mathbf{x})$ is k maximizing $p_k(\mathbf{x}, \boldsymbol{\theta})$.

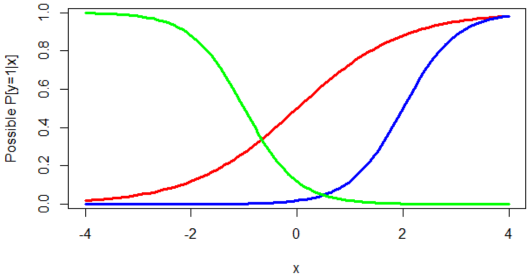
Logistic Regression

Implied Conditional Probabilities

Below is a plot of several different $p = 1$ forms for $p_1(x, \beta_0, \beta_1)$ in a $K = 2$ model. The parameter sets are

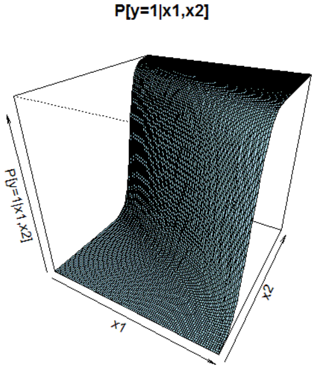
Red	$\beta_0 = 0, \beta_1 = 1$
Blue	$\beta_0 = -4, \beta_1 = 2$
Green	$\beta_0 = -2, \beta_1 = -2$

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 $|\beta_1|$.



Logistic Regression
Implied Conditional Probabilities

In a $K = 2$ case with $p = 2$, (for this $\{1, 2\}$ coding of y) the kind of relationship pictured below 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 $\beta_0 + \beta_1 x_1 + \beta_2 x_2 = c$, and takes the value .5 on the line $\beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0$.



Assumption (2) generalizes the "mixture of MVNs" assumption of LDA, and standard methods of fitting the corresponding parameters based on training data are necessarily fundamentally different. Using maximum likelihood in LDA, the K probabilities π_k, the K means $\boldsymbol{\mu}_k$, and the covariance matrix $\boldsymbol{\Sigma}$ are chosen to maximize the likelihood

$$\prod_{i=1}^{N} \pi_{y_i} g\left(\mathbf{x}_i | \boldsymbol{\mu}_{y_i}, \boldsymbol{\Sigma}\right)$$

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

$$\prod_{i=1}^{N} p_{y_i}\left(\mathbf{x}_i, \boldsymbol{\theta}\right) \qquad (3)$$

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, $1/N$ times the negative log-likelihood has a particularly simple form, namely

$$\frac{1}{N}\sum_{i=1}^{N}\ln\left[1+\exp\left(y_i\left(\beta_0+\sum_{j=1}^{p}\beta_j x_{ij}\right)\right)\right] \tag{4}$$

This is an empirical version of

$$\mathrm{E}\ln\left[1+\exp\left(-y\left(-\beta_0-\sum_{j=1}^{p}\beta_j x_j\right)\right)\right]$$

for the situation where $(\mathbf{x}, y) \sim P$. The arguments in Module 10 concerning voting functions show this is $\ln 2$ times an upper bound on the error rate of a classifier with voting function $-\beta_0 - \sum_{j=1}^{p}\beta_j x_j$. So coefficient vectors giving small values of (4) (i.e. large likelihood) can be expected to produce classifiers with small (0-1 loss) Err.

Logistic Regression

Penalized Likelihood Fitting

Optimization of (4) ignores the potential for overfitting. Penalization (for standardized inputs) of the logistic regression coefficients is a means of investigating a natural spectrum of fitted logistic regressions. For example, `glmnet` will optimize the elastic net penalized negative loglikelihood

$$\frac{1}{N}\sum_{i=1}^{N}\ln\left[1+\exp\left(y_i\left(\beta_0+\boldsymbol{\beta}'\mathbf{x}_i\right)\right)\right]+\lambda\left(\alpha\sum_{j=1}^{p}|\beta_j|+\frac{(1-\alpha)}{2}\sum_{j=1}^{p}\beta_j^2\right) \tag{5}$$

(where $\boldsymbol{\beta}\in\Re^p$). Comparison of **cross-validation** classification error rates across a grid of coefficient vectors (λ,α) affords appropriate choice of **complexity**.

Logistic Regression Example

Wisconsin Breast Cancer Example

The `train()` function in the `caret` package will use `glmnet` to search over optimizers of the elastic net penalized form (5), reporting the combination of penalty parameters with corresponding fit optimizing a cross-validation error for 0-1 loss. This was done using 100 repeats of 10-fold cross-validation. The resulting cross-validation error was 3.07%, somewhat better than what was obtained using $p = 9$ LDA.

The (`glmnet` form) parameters identified by cross-validation were roughly $\alpha = .005$ and $\lambda = .017$ and provide some shrinkage of coefficients (and thus flattening of the predicted probability surface) over what is produced by ordinary (unpenalized likelihood) logistic regression fitting.

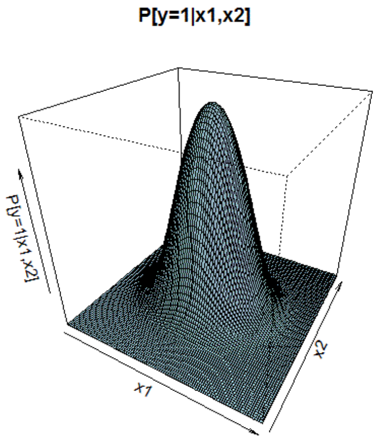
Logistic Regression
Basis Functions and Transforms

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 $x_1 x_2$, one can use logistic regression technology based on the 5-dimensional input $\left(x_1, x_2, x_1^2, x_2^2, x_1 x_2\right)$ 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 on the next slide.

Logistic Regression

Hypothetical Logistic Regression Probability that y=1

The plot below results when one uses the quadratic form $-.2x_1^2 - .3x_2^2$ to make logistic probabilities that $y = 1$ (for 1-2 coding). Constant-probability contours of such a surface are ellipses in (x_1, x_2)-space.



Statistical Machine Learning-13

Linear Methods of Classification Part 2: Support Vector Classifiers

Stephen Vardeman

Analytics Iowa LLC

April 2020

Other Linear Classifiers

Methods Not Derived from A Statistical Model

LDA and logistic regression provide classical statistical models as motivation for linear classifiers. There is another line of argument/ development that begins elsewhere, namely with a non-probabilistic classical optimization objective.

In the end, regardless of origins, all prediction and classification methodologies are subject to the hard scrutiny of how well they do at prediction/classification. So we next discuss this alternative line of development and what it offers in the way of new insights to and sound methods for classification.

Support Vector Classifiers

Goal in Vague Terms

Consider a 2-class classification problem. For notational convenience, we'll suppose that output y takes values in $\{-1, 1\}$. For $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_p) \in \Re^p$ and $\beta_0 \in \Re$ again consider the form

$$v(\mathbf{x}) = \beta_0 + \sum_{j=1}^{p} \beta_j x_j = \beta_0 + \mathbf{x}'\boldsymbol{\beta} \tag{1}$$

and a classifier

$$f(\mathbf{x}) = \text{sign}(v(\mathbf{x})) = \begin{cases} -1 & \text{if } v(\mathbf{x}) < 0 \\ 1 & \text{if } v(\mathbf{x}) > 0 \end{cases} \tag{2}$$

We consider the problem of choosing $\boldsymbol{\beta}$ and β_0 to provide a **"maximal cushion" around a hyperplane approximately separating between training inputs $\mathbf{x}_i$ with $y_i = -1$ and those $\mathbf{x}_i$ with $y_i = 1$.**

Support Vector Classifiers

Geometry of "Cushions Around Hyperplanes" in p-Space

A useful way to think about a $(p-1)$-dimensional hyperplane in $\Re^p$ is as the set of all $\mathbf{x} \in \Re^p$ that for a particular unit vector $\mathbf{u} \in \Re^p$ and constant c satisfy

$$\sum_{j=1}^{p} u_j x_j = c$$

Then for $M > 0$ the set of $\mathbf{x} \in \Re^p$ with

$$c - M \leq \sum_{j=1}^{p} u_j x_j \leq c + M$$

is a slab or cushion of thickness $2M$ around the hyperplane. Typically there is no such slab that separates the $y_i = -1$ input vectors $\mathbf{x}_i$ from the $y_i = 1$ input vectors $\mathbf{x}_i$ and one must settle for somewhat less than identification of a separating slab.

Support Vector Classifiers

Slack Variables

For some set of values $\xi_i \geq 0$ called "**slack**" variables (that provide "wiggle room" in search for a "thick" slab that "nearly separates" inputs for two classes) controlled for "**budget**" $C > 0$ by the constraint that

$$\sum_{i=1}^{N} \xi_i \leq C$$

we consider (for unit vector $\mathbf{u} \in \Re^p$ and constant β_0) the set of conditions

$$y_i \left(\beta_0 + \sum_{j=1}^{p} u_j x_{ij} \right) \geq M \left(1 - \xi_i \right) \ \forall i \qquad (3)$$

For $y_i = -1$ this is $\sum_{j=1}^{p} u_j x_{ij} \leq -\beta_0 - M \left(1 - \xi_i \right)$ and for $y_i = 1$ it is $\sum_{j=1}^{p} u_j x_{ij} \geq -\beta_0 + M \left(1 - \xi_i \right)$.

Support Vector Classifiers

Interpretation of Slack Variables

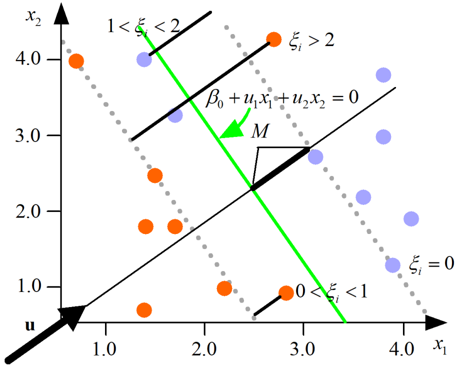
If a slab of thickness $2M$ around the hyperplane defined by $\sum_{j=1}^{p} u_j x_{ij} = -\beta_0$ separates the 2 classes, the conditions (3) are met with all $\xi_i = 0$. Requiring only that they all be met with some $\xi_i > 0$ is a relaxation of expectations. Here ξ_i is a fraction of the **margin** M that input $\mathbf{x}_i$ is allowed to be on the "wrong side" of its cushion around the hyperplane defined by $\sum_{j=1}^{p} u_j x_j = -\beta_0$.

The sign/direction on $\mathbf{u}$ is chosen to give the points with $y_i = 1$ larger $\sum_{j=i}^{p} x_{ij} u_j$ than the ones with $y_i = -1$.

Support Vector Classifiers

p=2 Graphical Representation of the Notation

Below is a toy $p = 2$ example illustrating the notation used regarding slack variables and a slabs meant to separate $y_i = -1$ (red) and $y_i = 1$ (blue) cases.



Support Vector Classifiers

Optimization Problem

The optimization problem corresponding to a support vector classifier (with $v(\mathbf{x}) = \beta_0 + \sum_{j=1}^{p} u_j x_j$) is to

$$\underset{\substack{\mathbf{u} \text{ with } \|\mathbf{u}\|=1 \\ \text{and } \beta_0 \in \Re}}{\text{maximize}} \quad M \text{ subject to } \begin{cases} y_i \left(\beta_0 + \sum_{j=1}^{p} u_j x_{ij} \right) \geq M(1-\xi_i) \quad \forall i \\ \text{for some } \xi_i \geq 0 \text{ with } \sum_{i=1}^{N} \xi_i \leq C \end{cases}$$

This constrained (by the budget $C > 0$) problem is equivalent for some $C^* > 0$ to the problem

$$\underset{\substack{\boldsymbol{\beta} \in \Re^p \\ \text{and } \beta_0 \in \Re}}{\text{minimize}} \quad \frac{1}{2}\|\boldsymbol{\beta}\|^2 + C^* \sum_{i=1}^{N} \xi_i \text{ subject to } \begin{cases} y_i \left(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} \right) + \xi_i \geq 1 \\ \forall i \text{ for some } \xi_i \geq 0 \end{cases}$$

(and use of $v(\mathbf{x}) = \beta_0 + \sum_{j=1}^{p} \beta_j x_j$).

Support Vector Classifiers

Second Version of the Optimization Problem

The second version of the problem replaces the budget constraint on $\sum_{i=1}^{N} \xi_i$ with a **penalty** on this sum, in much the same way that elastic net fitting can be described in terms of either a constrained optimization problem or a related unconstrained problem. This second formulation is that of a convex (quadratic criterion, linear inequality constraints) optimization problem for which there exists standard theory and algorithms.

Large C^* corresponds directly to a small budget constraint C in the first formulation. C^* (C) functions as a **complexity parameter** that can be chosen on the basis of **cross-validation**. Small C^* (large C) corresponds to a "low complexity" classifier and there are many support vectors contributing to the ultimate form of the classifier. The exact form of the classifier is less sensitive to a few key data cases when C^* is small than when C^* is large.

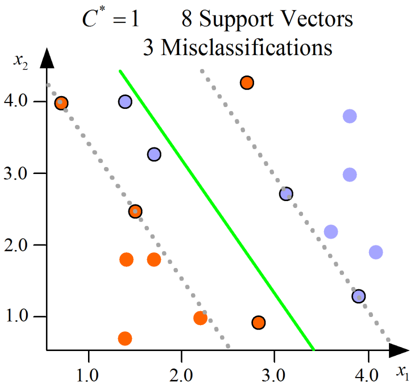
Support Vector Classifiers
Properties of the Optimal Solution

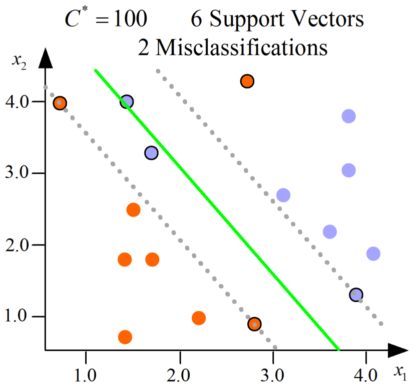
The optimizing vector $\boldsymbol{\beta}$ producing the optimal classifier has $\|\boldsymbol{\beta}\| = M^{-1}$ and can be written as $\sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$ for some $\alpha_i \geq 0$. In fact $\alpha_i > 0$ only for those (typically few) training cases for which $y_i \left(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} \right) \leq 1$. Geometrically these are the input vectors $\mathbf{x}_i$ that are "exactly on" (in the case of equality) or "on the wrong side of" the surface of the "cushion of thickness $2M$" that surrounds the hyperplane defining the decision boundary. They are called the **support vectors** of the classifier. The support vectors $\mathbf{x}_i$ with $y_i \left(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} \right) = 1$ have corresponding $\xi_i = 0$, while the others have $\xi_i > 0$. As C^* increases (C decreases) fewer and fewer support vectors are allowed.

Support Vector Classifiers

Two Toy Examples

Below are illustrations of two different support vector classifiers for a small $p = 2$ problem.





Support Vector Classifiers

Wisconsin Breast Cancer Example

The train() function in the caret package can be used to make a good cross-validation-based choice of C^* (in terms of 0-1 loss error rate) for the Wisconsin breast cancer example. 100 repetitions of 10-fold cross-validation produced a best average cross-validation error rate of 2.93% for the choice $C^* = .01$.

Subsequently using the ksvm() function in the kernlab package to fit a support vector classifier based on this cost value, a training error rate of 2.93% was also obtained. 11 out of 444 (2.48%) of benign training cases and 9 of 239 (3.77%) of malignant training cases are misclassified by the resulting support vector classifier. The classifier is potentially described as having low complexity, in that the small value of C^* produces a classifier that has 106 support vectors.

Statistical Machine Learning-14

Extensions of Linear Classifiers: New Features and Kernels (SVMs)

Stephen Vardeman

Analytics Iowa LLC

April 2019

Creation and Use of Functions of a p-Dimensional Input

We've already mentioned (in the contexts of LDA and logistic regression-based classifiers) the possibility of building from an input vector $\mathbf{x} \in \Re^p$ other predictors $h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_M(\mathbf{x})$ (some of which could be coordinates of $\mathbf{x}$) and doing classification based on the M-dimensional feature vector $(h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_M(\mathbf{x}))$. (Again, a version of "linear" classification with *quadratic* decision boundaries in $\Re^p$ can be obtained in this way.)

Of course, the same observation can be made about support vector classifiers. They can be directly used with features engineered from $\mathbf{x} \in \Re^p$. While this is possible, another more implicit route to the use of transformations/features is far more common with support vector classifiers. This is the use of "kernels" and the resulting "support vector machines."

Support Vector Classifiers and Kernels-SVMs

Kernel Functions

A key idea of modern machine learning is the use of "kernel" functions to create new non-linear methods from more standard linear ones. A kernel function (in this sense of the word "kernel" – the word is used a variety of ways in this field) is a symmetric function $K(\mathbf{x}, \mathbf{z})$ that maps pairs of inputs to real numbers in such a way that for any set of N inputs $\mathbf{x}_i$ the $N \times N$ matrix with i, i' entry $K(\mathbf{x}_i, \mathbf{x}_{i'})$, say $\mathbf{K}$, is "non-negative definite," meaning that for any N vector of values α_i,

$$\sum_{i,i'} \alpha_i \alpha_{i'} K(\mathbf{x}_i, \mathbf{x}_{i'}) \geq 0$$

(in vector notation, this is $\boldsymbol{\alpha}' \mathbf{K} \boldsymbol{\alpha} \geq 0$).

Roughly, one then uses N-vectors of "features" defined by the kernel and training set

$$(K(\cdot, \mathbf{x}_1), K(\cdot, \mathbf{x}_2), \ldots, K(\cdot, \mathbf{x}_N)) \tag{1}$$

to build a classifier (or other statistical learning method).

Not every function K with $K(\mathbf{x},\mathbf{z}) = K(\mathbf{z},\mathbf{x})$ is a kernel function. But there are many useful ones, among them the ordinary inner product in $\Re^p$ (the "linear kernel") and the so-called "Gaussian" kernel,

$$K_{\mathrm{l}}(\mathbf{x},\mathbf{z}) = \sum_{j=1}^{p} x_j z_j \quad \text{and}$$

$$K_{\mathrm{G}}(\mathbf{x},\mathbf{z}) = \exp\left(-\gamma \|\mathbf{x}-\mathbf{z}\|^2\right) = \exp\left(-\gamma \sum_{j=1}^{p} (x_j - z_j)^2\right)$$

respectively. **When a linear kernel is used, the corresponding machine learning method is exactly the "ordinary" method based on linear combinations of coordinates of** x. **And when a Gaussian kernel is used, a method based on linear combinations of "radial basis functions" located at training cases, functions** $\exp\left(-\gamma \|\mathbf{x}-\mathbf{x}_i\|^2\right)$**, is produced**.

Support Vector Classifiers and Kernels-SVMs

Inner Products Between "Kernel Features"

There is a theoretical basis to replace $\Re^N$ inner products of newly-produced "kernel features" with values of the kernel. That is, instead of using the $\Re^N$ inner product

$$\sum_{i=1}^{N} K(\mathbf{x}, \mathbf{x}_i) K(\mathbf{z}, \mathbf{x}_i)$$

for feature vectors (1) corresponding to inputs $\mathbf{x}$ and $\mathbf{z}$, "abstract inner products"

$$K(\mathbf{x}, \mathbf{z})$$

for these can be used in the development of classifiers (or other statistical learning methods). This is sometimes called "the kernel trick." Its real basis is theory that says there is an abstract (function) space with elements that are linear combinations of functions $K(\cdot, \mathbf{x})$ mapped to from the original feature space $\Re^p$ by $\mathbf{x} \longrightarrow K(\cdot, \mathbf{x})$ in which $\mathbf{x}$ and $\mathbf{z}$ are mapped to elements with inner product $K(\mathbf{x}, \mathbf{z})$.

Support Vector Machines (SVMs)

Application of Kernels to Support Vector Classification

Solving the support vector optimization problem for N-dimensional features (1) computing inner products using values of the kernel (or solving the support vector optimization problem in the abstract function space and translating the solution back to the original feature space $\Re^p$) produces "**support vector machines**."

Just as other linear classification methods applied directly to non-linear functions of $\mathbf{x}$ produce non-linear decision boundaries in $\Re^p$, use of any but the linear kernel $K_{\mathrm{l}}(\mathbf{x},\mathbf{z})$ produces non-linear decision boundaries (and contours in $\Re^p$ corresponding to the points in $\Re^N$ or the abstract space a "distance" in those spaces of "M" from the decision boundary in them). The numerical optimization required is exactly the same as that required to produce a support vector classifier ... so computationally, there is nothing any more difficult here than is already faced in making support vector classifiers.

Support Vector Machines (SVMs)

Form of an SVM

When $\mathbf{x} \in \Re^p$ a **support vector classifier** is for optimally chosen $\beta_0 \in \Re$ and $\boldsymbol{\beta} \in \Re^p$ an optimal linear combination of *some of* the training feature vectors $\mathbf{x}_i$ (the support vectors) is

$$\text{sign}\left(\beta_0 + \sum_{j=1}^{p} \beta_j x_j\right)$$

The form of a **support vector machine** is analogous. For $\mathbf{x} \in \Re^p$ and kernel function $K(\mathbf{x}, \mathbf{z})$, the analogue of $\boldsymbol{\beta} \in \Re^p$ is an optimally chosen linear combination of *some of* the training case kernel features (1) (the "support vectors" for the SVM). The analogue of $\sum_{j=1}^{p} \beta_j x_j$ is the (abstract) inner product of that linear combination with the kernel feature vector for $\mathbf{x}$. This is the same linear combination of the of the values $K(\mathbf{x}, \mathbf{x}_i)$. So for an optimally chosen $\beta_0 \in \Re$ and $\boldsymbol{\beta} \in \Re^N$, an SVM is

$$\text{sign}\left(\beta_0 + \sum_{i=1}^{N} \beta_i K(\mathbf{x}, \mathbf{x}_i)\right) \qquad (2)$$

Support Vector Machines (SVMs)

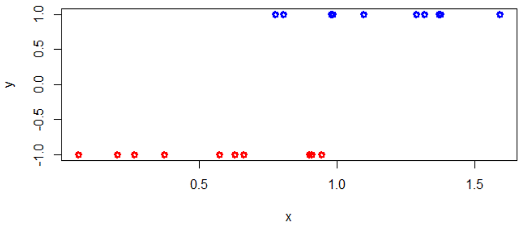
Form of an SVM

The form (2) is interesting. When the kernel is the Gaussian kernel, "voting" function $v(\mathbf{x}) = \beta_0 + \sum_{i=1}^{N} \beta_i K(\mathbf{x}, \mathbf{x}_i)$ is an element of a "radial basis function network" (is a linear combination of radial basis functions). The decision boundary in $\mathbf{x} \in \Re^p$ is the "0-level contour" of this function. "Support vectors" in this context are training cases that have non-zero α_i in the optimal form of the voting function. In general, large C^* produces (large penalty for slack variables in the abstract space and therefore relatively) small numbers of support vectors.

In this context, both the cost parameter C^* and the kernel parameter γ are **complexity parameters** whose values can be chosen via **cross-validation**.

Support Vector Machines (SVMs)
Some SVMs for a Toy p=1 Example

Below is a plot of a simple $N = 20$ point $p = 1$ training set with 10 cases with $y = -1$ and 10 with $y = 1$.



We fit several SVM classifiers to this data set using Gaussian kernels with parameter γ. "Voting functions" $\beta_0 + \sum_{i=1}^{N} \beta_i K(\mathbf{x}, \mathbf{x}_i)$ to be compared to 0 to produce classifications are thus "a constant plus a linear combination of normal pdfs with standard deviation $1/\sqrt{2\gamma}$". The next slide shows voting functions for 4 combinations of γ and the cost parameter C^*.

Support Vector Machines (SVMs)

SVM Voting Functions for a Toy p=1 Example

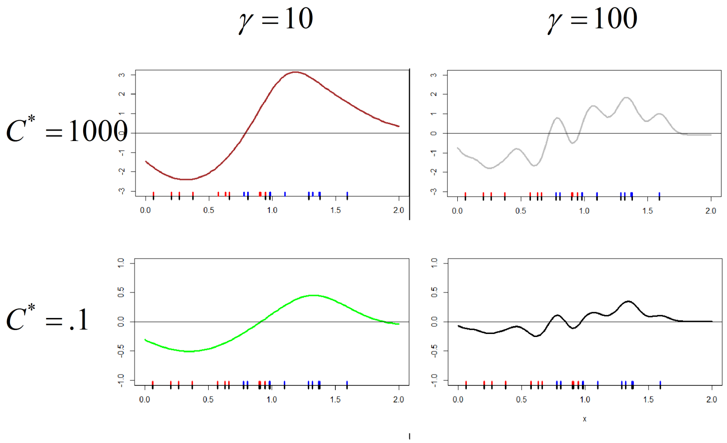


Figure: Red and blue ticks indicate locations of class -1 and 1 training cases. Ticks below the axes indicate locations of "support vector" cases.

Support Vector Machines (SVMs)

Observations Motivated by the Plots

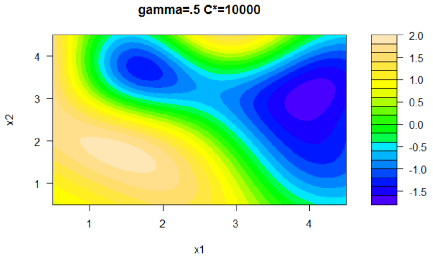
The plots on the previous panel suggest that

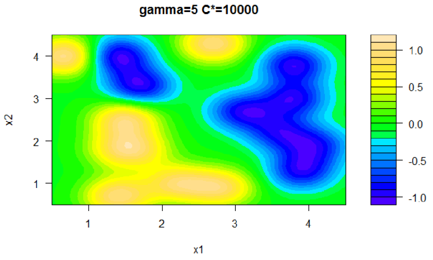
1. large γ enables "wiggly" voting functions (and the ability to match them to irregular patterns in training cases),
2. overall "amplitude" of the voting function increases with C^*, and
3. increased C^* reduces the number of support vectors.

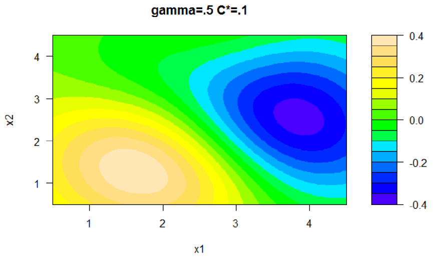
The next slide presents 4 contour plots of Gaussian kernel SVM voting functions for the $p = 2$ example used to illustrate (linear kernel) SV classifiers (laid out in the same pattern for relative sizes of γ and C^* just used in the $p = 1$ example). Qualitative similarities to the $p = 1$ example are clear.

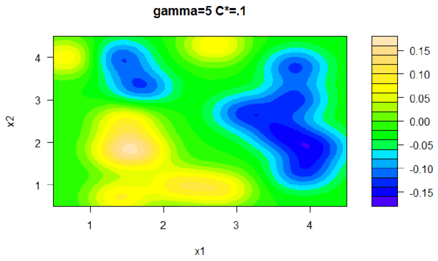
Support Vector Machines (SVMs)

A Toy p=2 Example









Support Vector Machines (SVMs)
Wisconsin Breast Cancer Example

It should be clear from even the toy examples that there is a huge variety of possibilities of fitted classifier that result from the choice of γ and C^* without even opening the possibility of other kernel forms. Choice of parameters for a real SVM is clearly a job for very careful cross-validation.

The `train()` function in the `caret` package can be used to make a good cross-validation-based choice of C^* and γ for an SVM with Gaussian kernel for the Wisconsin Breast Cancer data. 100 repetitions of 10-fold cross-validation produced a best average cross-validation error rate of 2.92% for the choice $C^* = 1.2$ with $\gamma = .05$. Subsequently using the `ksvm()` function in the `kernlab` package to fit a support vector classifier based on these values, a training error rate of .3% was obtained. 1 out of 444 benign training cases and 1 of 239 malignant training cases are misclassified by the resulting support vector machine that has 282 support vectors.

SVMs and Optimal Voting Functions

Perspective

SVM voting functions $v(\mathbf{x}) = \beta_0 + \sum_{i=1}^{N} \beta_i K(\mathbf{x}_i, \mathbf{x})$ optimize over choices of $\beta_0, \beta_1, \ldots, \beta_N$ an empirical version of (expected "hinge function-loss")

$$\mathrm{E}\left[1 - y \cdot v(\mathbf{x})\right]_+ \tag{3}$$

plus $1/C^*$ *times a penalty*[1] *on the "size" of the function* $\sum_{i=1}^{N} \beta_i K(\mathbf{x}_i, \mathbf{x})$. As indicated in Module 10, 0-1 loss error rate for $f(\mathbf{x}) = \text{sign}(v(\mathbf{x}))$ is bounded above by the expected hinge function-loss (3), and a g optimizing the expected hinge function-loss is

$$g^{\text{opt}}(\mathbf{x}) = \text{sign}\left(P\left[y = 1|\mathbf{x}\right] - \frac{1}{2}\right)$$

This is the optimal 0-1 loss classifier. So SVM technology produces a**n approximation of the optimal** 0-1 **loss classifier based on penalized linear combinations of kernel slices**.

[1] The penalty is the quadratic $\boldsymbol{\beta}'\mathbf{K}\boldsymbol{\beta}$ in $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_N)'$.

Statistical Machine Learning-15

Trees and Related Methods Part 2: Classification/Decision Trees and Random Forests for Classification

Stephen Vardeman

Analytics Iowa LLC

Janaury 2018

Classification/Decision Trees

K-Class Problem

The classification version of CART is similar to the SEL/regression version. An empirical loss to associate with a given tree (parallel to SSE used in regression contexts) is needed. Note that in a K-class problem (where y takes values in $\{1, 2, \ldots, K\}$) corresponding to a particular rectangle R_m is the fraction of training vectors with classification k,

$$\widehat{p_{mk}} = \frac{1}{\#\text{ training input vectors in } R_m} \sum_{i \text{ with } \mathbf{x}_i \text{ in } R_m} I\left[y_i = k\right]$$

and a natural classifier based on l rectangles is $\hat{f}_l(\mathbf{x})$ taking a value k maximizing $\widehat{p_{mk}}$ for the m such that $\mathbf{x} \in R_m$. That is, $\hat{f}_l(\mathbf{x})$ is a class most heavily represented in the rectangle to which $\mathbf{x}$ belongs. The empirical misclassification rate (0-1 loss training error) for this predictor is

$$\overline{\text{err}} = \frac{1}{N}\sum_{i=1}^{N} I\left[y_i \neq \hat{f}_l(\mathbf{x}_i)\right]$$

$\overline{\text{err}}$ is the most obvious splitting criterion for tree building (with splits chosen to provide maximum reduction in $\overline{\text{err}}$ available given a current set of rectangles). But other measures of overall node/rectangle purity (not directly tied to classification error rate) are also popular.

One such criterion is "the Gini index"

$$\frac{1}{N}\sum_{m=1}^{l} N_m \left(\sum_{k=1}^{K} \widehat{p_{mk}} \left(1 - \widehat{p_{mk}}\right) \right)$$

where $N_m =$ the number of training input vectors in R_m. In the $K = 2$ case, this is equivalent to SSE computed treating class labels as numerical values.

Classification/Decision Trees

Splitting Criteria and Choice of Tree

Another measure of overall node/rectangle purity is the "cross entropy"

$$-\frac{1}{N}\sum_{m=1}^{I} N_m \left(\sum_{k=1}^{K} \widehat{p_{mk}} \ln\left(\widehat{p_{mk}}\right)\right)$$

(The entropy of distribution $p_1, \ldots, p_K$ on K values is $-\sum_{k=1}^{K} p_k \ln(p_k)$, a number between 0 and $\ln K$, the former corresponding to the case where one $p_k = 1$ and the latter to the case where all $p_k = 1/K$.)

Upon adopting one of these criteria and using it 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.

Example

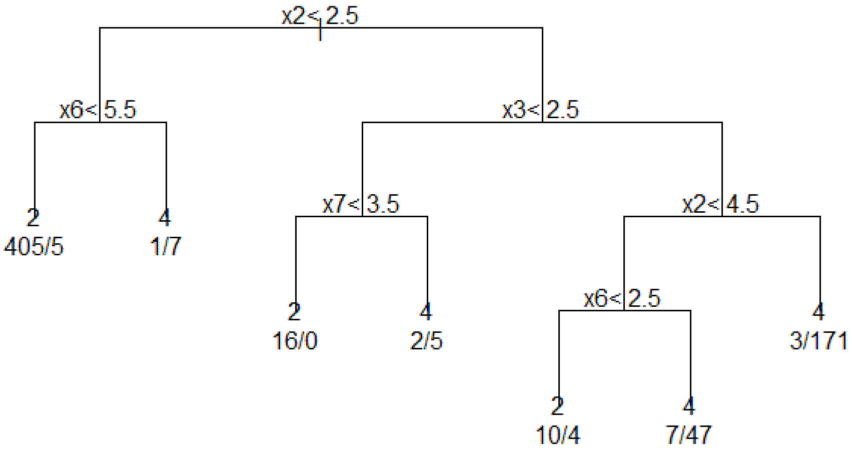
Wisconsin Breast Cancer Example

The `caret` function `train()` provides cross-validation choice of the `rpart` parameter c_p multiplying tree size in cost-complexity pruning of a decision tree for the Wisconsin Breast Cancer example. 10-fold cross-validation repeated 100 times yields a minimum 4.86% cross-validation error rate for the choice of $c_p = .0014$.

This value of c_p applied to making a decision tree based on the entire training set produces the very simple tree with 7 final leaves portrayed on the graphic on the next slide. The benign cases are coded "2" and the malignant ones are coded "4" on that graphic. It is clear from the graphic that 13 of 444 (2.93%) of benign and 9 of 239 (3.77%) of malignant training cases are misclassified by the decision tree, for an overall training error rate of 3.22%.

Example

Decision/Classification Tree for the Wisconsin Study



Random Forests for Classification

Decision/Classification Trees from Bootstrap Samples

As in the regression case, suppose that one makes B bootstrap samples of size N from the training set $\mathbf{T}$, say $\mathbf{T}_1^*, \mathbf{T}_2^*, \ldots, \mathbf{T}_B^*$. For each sample, $\mathbf{T}_b^*$, a corresponding classification tree is developed 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 (that most reduces the splitting criterion), 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}.

Let $\hat{f}^{*b}(\mathbf{x})$ be the corresponding tree-based classifier taking values in $\{1, 2, \ldots, K\}$, a class most heavily represented in the final rectangle to which $\mathbf{x}$ belongs.

Random Forests for Classification

A Random Forest Classifier

A random forest classifier is then $\hat{f}_B^*(\mathbf{x})$ a class k maximizing

$$\sum_{b=1}^{B} I\left[\hat{f}^{*b}(\mathbf{x}) = k\right]$$

the number of bootstrap trees classifying to k.

Standard default values of tuning/**complexity parameters** for random forest classifiers are $m = \lfloor \sqrt{p} \rfloor$ and $n_{\min} = 1$.[1] More principled choice of these parameters is possible via **cross-validation**.

[1] The default $n_{\min} = 1$ means that splitting terminates only because of reaching a maximum depth or the impossibility of reducing the splitting criterion with a single additional split. Unless the maximum tree depth is small, this default can easily lead to 0 training error rates that are not good indicators of Err. For that purpose the OOB error must be trusted.

Random Forests for Classification
Out of Bag Samples

As for SEL random forests, for each b call the set of OOB indices $I(b) \subset \{1, 2, \ldots, N\}$. ($I(b)$ is the set of indices for which the corresponding training vector does not get included in the bootstrap training set $\mathbf{T}_b^*$.) Then let $\hat{y}_{iB}^*$ be a k maximizing

$$\sum_{b \le B \text{ such that } i \in I(b)} I\left[\hat{f}^{*b}(\mathbf{x}_i) = k\right]$$

the number of bootstrap classification trees not built using case i that classify case i to class k. The out of bag error is then

$$OOB\left(\hat{f}_B^*\right) = \frac{1}{N} \sum_{i=1}^{N} I\left[y_i \neq \hat{y}_{iB}^*\right]$$

in 0-1 loss classification contexts. Large B values of this serve as natural approximations of 0-1 loss Err for a random forest classifier.

Example

Wisconsin Breast Cancer Example

The `randomForest()` function in the package by the same name can be used to search (on the basis of OOB error) over values of the parameter *mtry* (that specifies how many randomly chosen coordinates of the input vector are eligible for splitting at each stage of tree building). With the default value of $n_{\min} = 1$, the choice $mtry = 2$ produces minimum OOB misclassification error rate of 2.49%. That random forest classifier misclassifies 12 of 444 (2.70%) of benign and 6 of 239 (2.51%) of malignant training cases, for an overall training error rate of 2.64%.

Estimation of Class Probabilities

Use of Random Forests

There are contexts where a classification model is appropriate, but what is needed is not a classifier, but rather class probabilities conditioned on the input vector $\mathbf{x}$. One such case is where an AUC criterion is to be applied to judge 2-class prediction efficacy and an ordering of training cases is needed. Another is where an ensemble of methods is to be used to make a meta-classifier/super learner.

In these circumstances, a perfectly sensible procedure is to apply a SEL/regression random forest to 0-1 responses, $\tilde{y} = I[y = k]$, for each class k of interest. Another possibility is to build bootstrap trees (probably using a Gini index or cross entropy splitting criterion) then making vectors

$$\left(\hat{f}_1^{*b}(\mathbf{x}), \hat{f}_2^{*b}(\mathbf{x}), \ldots, \hat{f}_K^{*b}(\mathbf{x})\right)$$

that give relative frequencies of the bootstrap sample cases in the same rectangle as $\mathbf{x}$,. These can be averaged across B bootstrap samples to produce predictors for the vector of K values $P[y = k|\mathbf{x}]$.

Statistical Machine Learning-16

Boosting in Classifcation Problems

Stephen Vardeman

Analytics Iowa LLC

January 2018

Boosting and Classification

Generalities, AdaBoost.M1, and Current Possibilities

SEL boosting amounts to successively correcting predictors for a quantitative response y (that are each empirical approximations to a conditional mean function, $\mathrm{E}[y|\mathbf{x}]$) by repeatedly modeling and partially correcting for residuals from the current version of the predictor.

A natural question is "How might one apply the boosting idea to classification?" The so-called "AdaBoost.M1 algorithm" is one specific answer to that question, that actually predates the Friedman gradient boosting ideas.

In this module we begin by making some general comments about boosting and classification. Then we consider AdaBoost.M1. Finally, we end with some discussion of what is currently practically possible in this realm.

Boosting and Classification

Boosting and Voting Function Optimization

In light of our view of boosting in SEL prediction and consideration of (likelihood ratios and) voting functions as basic to classification problems, a natural path to application of boosting in classification is through **production of a voting function by successive approximation to an optimizer of (an empirical version of) an expected function-loss by linear combination of elements of some class of simple functions of an input $\mathbf{x} \in \Re^p$.**

We first considered function-losses $h(u)$ for 2-class classification voting functions $v(\mathbf{x})$ in Module 10. We've since seen the usefulness of $h(u) = \ln(1 + \exp(-u)) / \ln(2)$ and coordinate functions of $\mathbf{x}$ in logistic regression-based classification and the relevance of the hinge loss $h(u) = (1 - u)_+$ (and kernel-slice functions) to support vector machines. This module further advances this story line.

Boosting and Classification

Boosting and Voting Function Optimization

In principle, any convenient function-loss $h(u)$ could be paired with any convenient class of functions of $\mathbf{x} \in \Re^p$ to motivate the development of a classification methodology. What is needed for effective application is good fitting methodology (penalized or not, one-shot or iterative) to produce (?approximate?) optimizers of the empirical mean function-loss. (In a penalized context, a spectrum of approximate optimizers can be produced.)

Although the original motivation of AdaBoost.M1 was not this, it can be seen as the pairing of **exponential function-loss** $h(u) = \exp(-u)$ with the **class of "stumps"**/single-split trees. It is an **iterative**/successive approximation algorithm for **unpenalized optimization** of empirical average function-loss.

The AdaBoost.M1 Algorithm

Consider again the $\{-1,1\}$ coding of $K=2$ class labels in a 0-1 loss classification problem (y takes values in $\{-1,1\}$). A voting function for classification that optimizes

$$\mathrm{E}\exp\left(-y\cdot v\left(\mathbf{x}\right)\right)$$

not only guarantees a upper bound on 0-1 loss error rate, but is in fact an minimizer of that error rate. The AdaBoost.M1 algorithm aims to optimize an empirical approximation of this, namely

$$\frac{1}{N}\sum_{i=1}^{N}\exp\left(-y_i\hat{f}\left(\mathbf{x}_i\right)\right)$$

by successive correction of linear combinations of the simplest possible tree classifiers–ones each splitting the $\mathbf{x}$ space on only one coordinate x_j–taking values ± 1.

The AdaBoost.M1 Algorithm

The AdaBoost.M1 algorithm makes exactly optimal updates (no gradient search for approximately optimal updates is needed) of successive approximations to an optimizer of empirical average function-loss. The class of linear combinations of "stumps" taking values -1 and 1 is large enough that the algorithm eventually produces an iterate with associated classifier that has the smallest training error rate possible. Except in situations where some training cases have $\mathbf{x}_i = \mathbf{x}_{i'}$ but $y_i \neq y_{i'}$ this best possible training error rate is 0.

As there is no formal penalization in the AdaBoost.M1 algorithm, the number of iterations that it is allowed to proceed is often treated as a **complexity parameter** and subjected to optimization based on **cross-validation** in order to address the issue of overfit.

The AdaBoost.M1 Algorithm

Algorithm Details

The AdaBoost.M1 algorithm is built on some base classifier form f (typically "stumps"–the simplest possible tree classifier, possessing just 2 final nodes). It proceeds as follows.

1. Initialize weights on the training data $(\mathbf{x}_i, y_i)$ at

$$w_{i1} \equiv \frac{1}{N} \text{ for } i = 1, 2, \ldots, N$$

2. Fit a classifier $\hat{f}_1$ to the training data to optimize

$$\sum_{i=1}^{N} I\left[y_i \neq \hat{f}\left(\mathbf{x}_i\right)\right]$$

let

$$\overline{\text{err}}_1 = \frac{1}{N} \sum_{i=1}^{N} I\left[y_i \neq \hat{f}_1\left(\mathbf{x}_i\right)\right] \quad \text{and} \quad \alpha_1 = \ln\left(\frac{1 - \overline{\text{err}}_1}{\overline{\text{err}}_1}\right)$$

The AdaBoost.M1 Algorithm

Algorithm Details

3. Set new weights on the training data

$$w_{i2} = \frac{1}{N} \exp\left(\alpha_1 I\left[y_i \neq \hat{f}_1(\mathbf{x}_i)\right]\right) \quad \text{for } i = 1, 2, \ldots, N$$

   (This up-weights misclassified cases by a factor of $\left(1 - \overline{\text{err}}_1\right) / \overline{\text{err}}_1$).)

4. For $m = 2, 3, \ldots, M$

   a. Fit a classifier $\hat{f}_m$ to the training data to optimize

$$\sum_{i=1}^{N} w_{im} I\left[y_i \neq \hat{f}(\mathbf{x}_i)\right]$$

   b. Let

$$\overline{\text{err}}_m = \frac{\sum_{i=1}^{N} w_{im} I\left[y_i \neq \hat{f}_m(\mathbf{x}_i)\right]}{\sum_{i=1}^{N} w_{im}} \quad \text{and} \quad \alpha_m = \ln\left(\frac{1 - \overline{\text{err}}_m}{\overline{\text{err}}_m}\right)$$

The AdaBoost.M1 Algorithm

Algorithm Details

c. Update weights as

$$w_{i(m+1)} = w_{im} \exp\left(\alpha_m I\left[y_i \neq \hat{f}_m(\mathbf{x}_i)\right]\right) \quad \text{for } i = 1, 2, \ldots, N$$

5. Output a resulting voting function

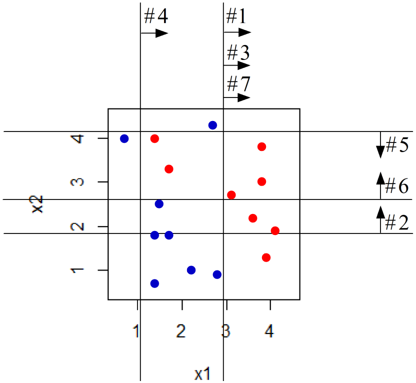
$$v_M(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m \hat{f}_m(\mathbf{x})$$

(Classifiers with small $\overline{\text{err}}_m$ get big positive weights.)

The AdaBoost.M1 Algorithm

Toy Example

Below is a graphic of the small ($N = 16$) fake $p = 2$ data set that has been used to illustrate the development of various classifiers, with (single line) boundaries of $M = 7$ successive "stumps" used to develop an AdaBoost classifier with 0 training error rate. (Arrows point in the direction of $y = +1$ decisions.)



The AdaBoost.M1 Algorithm

The Toy Example Continued

The first cut leaves 2 (red) cases with $y = 1$ on the wrong side of an $M = 1$ decision boundary. The first classifier thus has error rate $\overline{\text{err}}_1 = 2/16$ and $\alpha_1 = \ln\left((7/8)\,/\,(1/8)\right) = \ln(7)$. The two points on the wrong side of the decision boundary are up-weighted by a factor of 7 in seeking the next stump. One operates as if there are $14 + 7 + 7 = 28$ cases (7 at each of the $\mathbf{x}_i$ that are misclassified at the first cut) in choosing it. This leads to the #2 (horizontal line) decision boundary. And so on. At every iteration m, cases misclassified by the new stump are up-weighted (producing a new total weight). Ultimately, ($-1/1$ valued) stump classifiers $\hat{f}_m$ are combined via

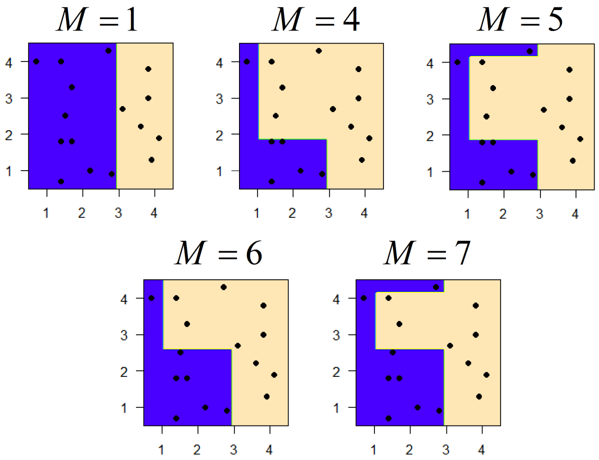
$$\sum_{m=1}^{M} \alpha_m \hat{f}_m(\mathbf{x})$$

to make the AdaBoost.M1 voting function. For instance, the first two terms of this voting function are $\ln(7)\text{sign}(x_1 - 2.95)$ and $\ln(6)\text{sign}(x_2 - 1.85)$.

The AdaBoost.M1 Algorithm

The Toy Example Continued

Below are graphics portraying the 5 different classifiers met in the development of the 0 training error rate $M = 7$ AdaBoostM.1 classifier.



AdaBoost

Wisconsin Breast Cancer Example

The `tune()` function in the `caret` package enables cross-validation for both the `adaboost()` routine in the `FastAdaboost` package and the `adaboost.M1()` routine in the `adabag` package. AdaBoost is extremely computationally intensive. The former implementation (in `C++` underneath an `R` interface) purports to be about 50 times as fast as the second. It seems that this is necessary for a problem of any size (despite the fact that the `adabag` package provides more options than the `FastAdaboost` implementation–including trees deeper than a single split) if any serious cross-validation is to be employed to tune the algorithm or simply predict its performance with a fixed set of parameters.

We tuned the `adaboost()` routine using 10 repeats of 10-fold cross-validation on the number of "iterations," i.e. on the parameter M. $M = 40$ stumps gave an average cross-validation error rate of 2.72%. AdaBoost.M1 run with $M = 40$ on the full training set has a 0 training error rate.

Boosting in Current Practice

Perspective

The currently wildly popular **XGBoost** algorithm implements **gradient boosting with general binary trees** as base classifiers (not just "stumps"). It allows not only "standard" choices of **empirical losses** (in a way that includes both regression-type prediction and classification) but **user-supplied** ones as well. It also provides the additional feature of employing a form of **penalization** to control the complexity of (tree) corrections added into the evolving form of a function being constructed. XGBoost is widely used in predictive analytics competitions. It is implemented in a **computationally very effective** manner, making **cross-validation tuning** workable (even without the necessity of access to a super-computer!)

Statistical Machine Learning-17

Combining Ensembles of Classifiers

Stephen Vardeman

Analytics Iowa LLC

January 2018

Combinations of "Ensembles" of Classifiers

Classifier "Fusion"

Often a person or team will create several classifiers, no one of which is completely satisfactory. A reasonable question is then "Is there any rational way to 'stack' or combine the ensemble of classifiers?" There are simple (linear combination) ways of combining SEL predictors that are reasonably defensible, but exactly what to do with multiple classifiers is far more confused. **While there is a fairly large literature on "classifier fusion," it is not cogent and in fact is rife with errors (of perspective and motivation if not technical detail)**. We here make some simple points about the problem and some general recommendations about what can be done.

Classifiers as "Features"

Likelihood Ratios for Classifier Outcomes as Data

In one sense, a classifier or (in the case that there is one, its underlying "voting function") is just one highly data-dependent "feature" computed from training data. If one ignores the data-dependent nature of these features (as is common in the "classifier fusion" literature) how best to combine them is completely obvious (**and ignored in the literature!**). For p classifiers **with vector of outcomes** (be they classifications or values of voting functions) $\mathbf{x} \in \Re^p$, in a $K = 2$ class problem with class-conditional densities (**for the vector of outcomes**) g_0 and g_1 and $\pi_0 = P[y = 0]$ and $\pi_1 = P[y = 1]$ an optimal classifier is

$$f(\mathbf{x}) = I\left[\frac{g_1(\mathbf{x})}{g_0(\mathbf{x})} > \frac{\pi_0}{\pi_1}\right]$$

One simply makes likelihood ratios based on the classifier outcome "data" $\mathbf{x}$ and classifies accordingly!

Optimal "Fusion" of Fixed Classifiers

A Toy Example

Consider the "fusion" of two classifiers, f_1 and f_2. Suppose that class-conditional joint probabilities for values of (f_1, f_2) are as below:

$y = 0$

	$f_2 = 0$	$f_2 = 1$
$f_1 = 1$	.45	.05
$f_1 = 0$	.45	.05

$y = 1$

	$f_2 = 0$	$f_2 = 1$
$f_1 = 1$	.05	.05
$f_1 = 0$	.45	.45

Taking ratios of cell probabilities, the likelihood ratio $g_1(f_1, f_2)/g_0(f_1, f_2)$ based on these classifiers is as below:

	$f_2 = 0$	$f_2 = 1$
$f_1 = 1$	$1/9$	1
$f_1 = 0$	1	9

Theoretically Optimal "Fusion" of Fixed Classifiers

Toy Example Continued

Then supposing that $\pi_0 = .6$ and $\pi_1 = .4$, an optimal classifier to be made from the pair (f_1, f_2) is

$$I\left[(f_1, f_2) = (0, 1)\right]$$

It is widely and wrongly assumed that a sensible method of combining multiple classifiers is always to "take a majority vote." This (admittedly artificial but instructive) example shows that heuristic can be seriously flawed (as it would demand deciding for class 1 when $(f_1, f_2) = (1, 1)$ if it decides for class 1 when $(f_1, f_2) = (0, 1)$). What matters is not numbers out of p classifiers "voting" for a class, but rather how joint probabilities for an observed vector of classification outcomes compares class to class. In a K-class problem with p classifiers producing outcome vector $\mathbf{x} \in \Re^p$, one wants to classify to class k maximizing

$$\pi_k g_k(\mathbf{x})$$

across $k = 1, 2, \ldots .K$. This need *not* have any simpler description.

"Linear Combination Stacking" versus "Meta-Classifiers"/"Super-Learners"

Common Heuristics versus Super-Learners

Majority voting is the simplest version of the common heuristic of making some set of weights $w_1, w_2, \ldots, w_p$ for 0-1 outcomes from p classifiers and using w-weighted voting to determine an ensemble classification. **But none of this naive kind of "linear combination stacking" has any real basis or motivation in theory.**

What is far more defensible is to **treat empirically-derived voting functions (or monotone transforms of them like estimates of $P\left[y = 1|\mathbf{x}\right]$ or approximate likelihood ratios) as features or inputs (possibly alongside some or all original inputs x_j) into some single tree-based classification method**. We say tree-based because of the invariance of such methods to scales of their inputs, obviating any need to somehow put all constituent voting functions on the same scale before applying a classification technology.

Super-Learners/Use of (a Final) Classifier Methodology

Voting Functions as Features

If a 2-class classification method is any good, it ultimately creates a partition of $\Re^p$ approximating the optimal one derived from the likelihood ratio. That means that any good voting function (or equivalent function) defining a classifier in an ensemble is approximately equivalent to the likelihood ratio. Using such as features in a tree-based classification method is then **simply a sensible way of hopefully building a better approximate likelihood ratio from the members of the ensemble**.

The obvious candidates for a final/"meta-classification" method are classification trees, random forests, and tree-based versions of boosting.

Super-Learners/Use of (a Final) Classifier Methodology

Honest Cross-Validation

The practical difficulties faced in principled application of the super-learner/meta-classifier paradigm are exactly those discussed in Module 9 concerning stacking/super-learner technology in SEL prediction. Honest cross-validation is the only means available for judging the likely effectiveness of a set of parameters for constituent classifiers and a final classifier. All of the discussion there about cross-validation in the context of SEL prediction super-learners is equally applicable here in the classification context.

Statistical Machine Learning-18

Principal Components

Stephen Vardeman

Analytics Iowa LLC

January 2018

Unsupervised Statistical Machine Learning

Interpretable Patterns Without the Goal of Prediction

All of statistical machine learning is about quantifying interpretable patterns in large numbers of variables and/or cases. We've looked extensively at "supervised learning" (prediction and classification). Now we consider some basic methods of "**unsupervised learning**" where there is no single target, y, in sight and the object is to make general statements about how variables or cases in an $N \times p$ training set are related.

We first look at "**principal components**" methods. These provide a "change of coordinate system" from the original p variables x_j, in which large (or small) variation in a particular new coordinate can be identified, potentially interpreted, and possibly provide a low-dimensional representation of high-dimensional (large p) data.

Matrices of Centered (and Possibly Scaled) Columns

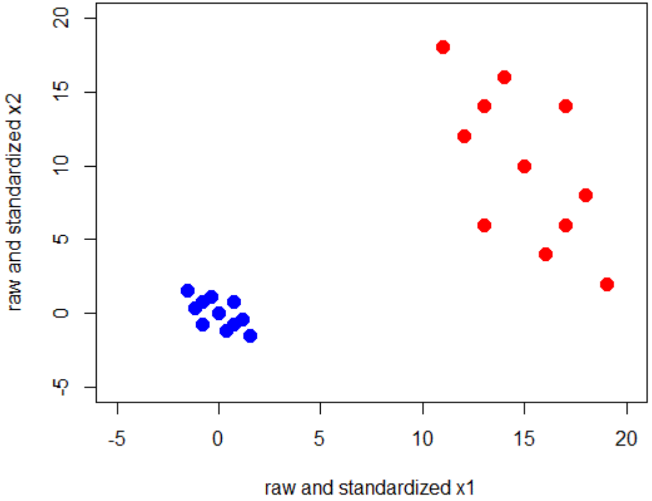
Relationships Among Variables

We assume that variables in an $N \times p$ training set have been centered, and possibly standardized by division by column standard deviations, s_j. This means that the "gross features" of overall averages–and possibly unit-dependent scale–are understood and have been removed from consideration. What remains is to understand how the set of variables are related.

This basic idea is pictured on the next slide. The $N = 11$ original (red) $p = 2$ data points have been standardized to produce the (blue) points. The original means of $\bar{x}_1 = 10$ and $\bar{x}_2 = 15$ and standard deviations $s_1 = 2.61$ and $s_2 = 5.22$ give way to 0 means and standard deviations of 1 for the standardized variables. But the correlation of $r = -.65$ is unchanged as one quantification of the basic relationship between the variables that remains.

Matrices of Centered (and Possibly Scaled) Columns

Relationships Among Variables



Matrices of Centered Columns

Application of the SVD (or Eigen Decompositions)

Applying the "singular value decomposition" of the $N \times p$ training data matrix $\mathbf{X}$ or the related "spectral/eigen decompositions" of matrices of inner products of rows or of columns of $\mathbf{X}$ (the latter giving a multiple of the sample covariance matrix for $\mathbf{X}$) identifies a new (rotated) coordinate system for p-dimensional space that aligns with the dataset in a way that identifies directions of important variation or lack thereof in the cases.

That is, appropriate linear algebra (theory and computational routines) produces a set of perpendicular unit[1] vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_p$ that define a new coordinate system in $\Re^p$ aligned with directions of "largest through smallest variation" in the the training set. These are called "**principal component directions**."

[1] The sum of the squared entries of any $\mathbf{v}_j$ is 1, making it interpretable as pointing from the origin of $\Re^p$ to a point on the unit sphere in $\Re^p$.

Matrices of Centered Columns
Application of the SVD

To be more precise, the singular value decomposition provides a unit vector $\mathbf{v}_1$ such that (for a matrix of centered columns) the sum of squared inner products with data cases, namely

$$\sum_{i=1}^{N}\left(\sum_{j=1}^{p} v_{1j}x_{ij}\right)^2$$

is maximal. This is N (or $N-1$) times the sample variance of the values $\sum_{j=1}^{p} v_{1j}x_{ij}$. These inner products are "1st coordinates" of case i data vectors in a new coordinate system defined by the principal component directions. $\mathbf{v}_1$ points in a direction of maximal-data-case-variation in terms of a new "$\mathbf{v}_1$ coordinate."

Subject to being perpendicular to (having 0 inner product with) $\mathbf{v}_1$, $\mathbf{v}_2$ points in an $\Re^p$ direction with maximal sample variance of the values $\sum_{j=1}^{p} v_{2j}x_{ij}$, the "2nd coordinates" of case i data vectors in a new coordinate system." And so on.

More Matrices of Centered Columns

Terminology

The inner products

$$\sum_{j=1}^{p} v_{lj} x_{ij} = \langle \mathbf{v}_l, \mathbf{x}_i \rangle$$

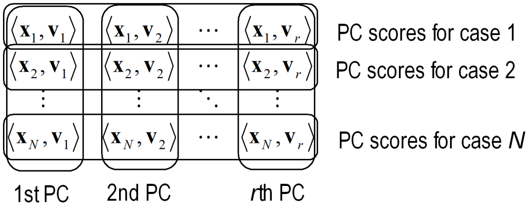
that are (the lth) coordinates of case i in the new coordinate system are called the **principal component scores** of the $\mathbf{x}_i$, and an N-vector consisting of the $\langle \mathbf{x}_i, \mathbf{v}_l \rangle$ (the new lth coordinates of the cases), is the value of the lth **principal component.**

The entries of the $p \times 1$ vector $\mathbf{v}_l$ are sometimes called the **component weights** or **loadings** for the lth component. A 0 loading means that the lth coordinate of $\mathbf{x}_i$ is ignored in the creation of its lth principal component score (the lth column of the data matrix does not participate in the creation of the lth principal component vector as a linear combination of columns of the data matrix $\mathbf{X}$).

More Matrices of Centered Columns

Terminology Summary

Here is a summary of the notation/terminology of principal components.

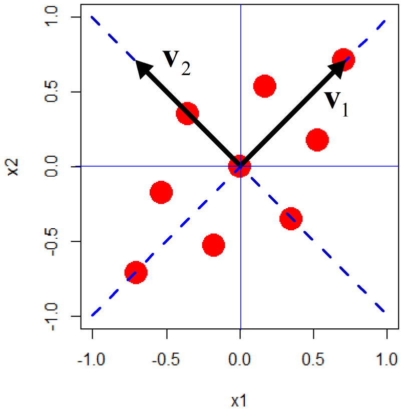

$\langle \mathbf{x}_1, \mathbf{v}_1 \rangle$ $\langle \mathbf{x}_1, \mathbf{v}_2 \rangle$ $\cdots$ $\langle \mathbf{x}_1, \mathbf{v}_r \rangle$
$\langle \mathbf{x}_2, \mathbf{v}_1 \rangle$ $\langle \mathbf{x}_2, \mathbf{v}_2 \rangle$ $\cdots$ $\langle \mathbf{x}_2, \mathbf{v}_r \rangle$
$\vdots$ $\vdots$ $\ddots$ $\vdots$
$\langle \mathbf{x}_N, \mathbf{v}_1 \rangle$ $\langle \mathbf{x}_N, \mathbf{v}_2 \rangle$ $\cdots$ $\langle \mathbf{x}_N, \mathbf{v}_r \rangle$
PC scores for case 1
PC scores for case 2
PC scores for case N
1st PC
2nd PC
rth PC

$$\mathbf{v}_j = \begin{array}{cl} \left(\begin{array}{c} v_{j1} \\ v_{j2} \\ \vdots \\ v_{jp} \end{array} \right) & \begin{array}{l} \text{PC } j \text{ loading on variable 1} \\ \text{PC } j \text{ loading on variable 2} \\ \\ \text{PC } j \text{ loading on variable } p \end{array} \end{array}$$

Matrices of Centered Columns

A Small p=2 Example

Below is a small $p = 2$ data set, with principal component direction vectors and corresponding new coordinate system shown (dashed blue).



Case	x_1	x_2	1st PC Score	2nd PC Score
1	-.707	-.707	-1.0	0.00
2	-.177	-.530	-0.5	-0.25
3	-.530	-.177	-0.5	0.25
4	.354	-.354	0.0	-0.50
5	.000	.000	0.0	0.00
6	-.354	.354	0.0	0.50
7	.530	.177	0.5	-0.25
8	.177	.530	0.5	0.25
9	.707	.707	1.0	0.00

Principal Components

Low-Dimensional Approximations to X

Best "low-dimensional" approximations to the data matrix $\mathbf{X}$ can be made using principal component directions and scores. A dataset that replaces case i with

$$(\text{case } i \text{ first principal component score}) \cdot \mathbf{v}_1$$

is the best possible 1-dimensional approximation to $\mathbf{X}$ (in terms of sum of squared differences between elements of $\mathbf{X}$ and the approximation). A data set that replaces case i with

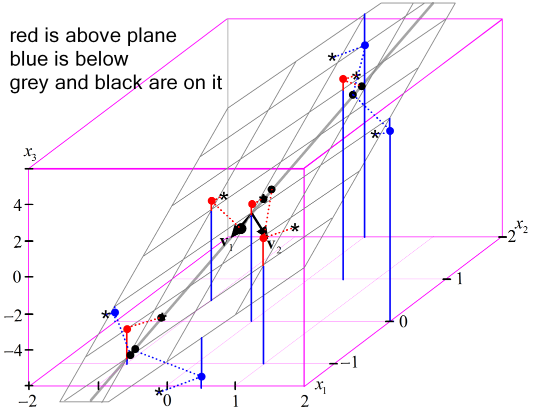
$$\sum_{l=1}^{L} (\text{case } i \text{ principal component score } l) \cdot \mathbf{v}_l \qquad (1)$$

is the best possible L-dimensional approximation to $\mathbf{X}$.

Principal Components

Example Geometry of Low-Dimensional Approximations to X

Below is a representation of a $p = 3$ data set. Shown are the $N = 9$ data points, its best 1-D approximation (black balls on the *line* defined by the first PC direction) and its best 2-D approximation (black stars on the *plane*).



Principal Components

Singular Value Decomposition and Eigen Analyses

Associated with principal component directions are "**singular values**." These are non-negative values d_l that decrease as the index on the principal component direction $\mathbf{v}_l$ increases. The sum of the squared entries of the L-dimensional approximation to $\mathbf{X}$ in display (1) is $\sum_{l=1}^{L} d_l^2$. So these are related to the portion of the variance of the (centered) values in $\mathbf{X}$ accounted for by the first L principal components.

The principal component directions are also so-called **eigenvectors** of the $p \times p$ matrix of inner products for columns of $\mathbf{X}$, and the singular values are the square roots of the so-called **eigenvalues** of that matrix. (Those are in turn a multiple of the eigenvalues for the sample covariance or correlation matrices for $\mathbf{X}$). Principal components analysis is then sometimes based not on the singular value decomposition of $\mathbf{X}$, but on an eigen analysis of a covariance or correlation matrix.

Principal Components

Some Uses

Principal components analyses are more or less helpful depending upon the profile in the singular values/eigenvalues. Where the d_l are roughly all of the same order of magnitude $l = 1, 2, \ldots, p$, a principal components analysis provides little insight into the structure of a training set. But where there are a few large or small singular values, one has potentially learned something valuable.

Where a few large singular values dominate, a few variables that are linear combinations of the original variables (namely the first few linear combinations $\sum_{j=1}^{p} v_{lj} x_j$) account for most of the variation seen in $\mathbf{X}$. Sometimes attempts are then made to interpret the loadings v_{lj} (particularly their signs) for a given l, hoping to see the principal components as natural averages of and contrasts between input variables.

Principal Components
More on Uses

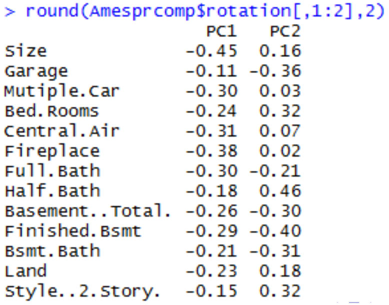
Where a few singular values dominate, cases that more or less match on the corresponding principal component scores are close together on a low-dimensional approximation of the data set, and may be more easily identified than in a full p-dimensional look at the training set. And this **dimension-reduction** view of principal components suggests their use in large p prediction problems where one might either use them in step-wise forward-selection fashion (thereby employing "principal components regression"), or might replace some group of predictors with a few of their principal components in development of a predictor.

Where a few singular values are extremely small, they point out **effective linear dependencies** among the x_js. Particularly where p variables x_j are actually transforms of a much smaller number of predictors, knowing that very late principal components are essentially 0 can provide insight about relationships among that small number of predictors.

Principal Components

Ames Housing Example

In R, one can either center (and potentially scale) an $N \times p$ data set and apply the `svd()` function, or directly apply the `prcomp()` function to produce principal component directions. (The `princomp()` function finds these via eigen analysis of the sample covariance or correlation matrix.) Considering only the $p = 13$ Ames house price predictors (and not the price variable) and standardizing before finding directions, below are the loadings for the first 2 principal component directions.



```
> round(Amesprcomp$rotation[,1:2],2)
                      PC1   PC2
Size                -0.45  0.16
Garage              -0.11 -0.36
Mutiple.Car         -0.30  0.03
Bed.Rooms           -0.24  0.32
Central.Air         -0.31  0.07
Fireplace           -0.38  0.02
Full.Bath           -0.30 -0.21
Half.Bath           -0.18  0.46
Basement..Total.    -0.26 -0.30
Finished.Bsmt       -0.29 -0.40
Bsmt.Bath           -0.21 -0.31
Land                -0.23  0.18
Style..2.Story.     -0.15  0.32
```

Statistical Machine Learning-19

Clustering

Stephen Vardeman

Analytics Iowa LLC

April 2020

Clustering
Generalities

A second form of unsupervised learning methodology "**clustering**." Typically the object in clustering is to find natural groups of rows or columns[1] of

$$\underset{N\times p}{\mathbf{X}} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{Np} \end{pmatrix}$$

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, let $\mathbf{x}_i$ $i = 1, 2, \ldots, r$ stand for **"items" to be clustered** (that might–but need not–be rows or columns of $\mathbf{X}$ with entries that need not be continuous variables).

[1]In some contexts one may want to somehow find homogenous "blocks" in a properly rearranged $\mathbf{X}$ and "bi-clustering" is appropriate.

Clustering
Generalities

Clustering methods are then built on some dissimilarity measure $d(\mathbf{x},\mathbf{z})$ that (at least for the items to be clustered and perhaps for other possible items) quantifies how "unalike" 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. $d(\mathbf{x},\mathbf{z}) \leq d(\mathbf{x},\mathbf{w}) + d(\mathbf{z},\mathbf{w}) \ \forall \mathbf{x},\mathbf{z}$.

Where all of 1-4 hold, d is a "metric."

Clustering
Generalities

If rows of $\mathbf{X}$ are being clustered and each column of $\mathbf{X}$ contains values of a continuous variable, a squared Euclidean distance is a natural choice for a dissimilarity measure

$$d\left(\mathbf{x}_i, \mathbf{x}_{i'}\right) = \left\|\mathbf{x}_i - \mathbf{x}_{i'}\right\|^2 = \sum_{j=1}^{p} \left(x_{ij} - x_{i'j}\right)^2$$

If columns of $\mathbf{X}$ are being clustered 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

$$d\left(\mathbf{x}_j, \mathbf{x}_{j'}\right) = 1 - \left|r_{jj'}\right|$$

Clustering
Generalities

When dissimilarities between pairs of 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 the information about items needed in order to cluster them. The objective of clustering is a partitioning of the index set $\{1, 2, \ldots, r\}$ 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.

Partitioning Methods ("Centroid"-Based Methods)

Generalities

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 $\mathbf{X}$ for cases where the columns contain values of continuous variables x_j. The version of the method we will describe uses squared Euclidean distance to measure dissimilarity.

In light of the fact that the algorithm is distance-based, to remove dependence of what it produces on units that are used, it will typically be appropriate to standardize columns of $\mathbf{X}$ before beginning.

K-Means Algorithm (A Centroid-Based Method)

First Iteration of the K-Means Algorithm

The algorithm then begins with some set of K distinct "centers" $\mathbf{c}_1^0, \mathbf{c}_2^0, \ldots, \mathbf{c}_K^0$. They might, for example, be a random selection of the rows of $\mathbf{X}$. Each $\mathbf{x}_i$ is assigned to that center $\mathbf{c}_{k^0(i)}^0$ minimizing

$$d\left(\mathbf{x}_i, \mathbf{c}_l^0\right)$$

over choice of l (creating K clusters around the centers). Next, each $\mathbf{c}_k^0$ is replaced with the corresponding cluster mean

$$\mathbf{c}_k^1 = \frac{1}{\#\text{ of } i \text{ with } k^0(i) = k} \sum I\left[k^0(i) = k\right] \mathbf{x}_i$$

K-Means Algorithm
mth Iteration of the Algorithm

At stage m with all $\mathbf{c}_k^{m-1}$ available, each $\mathbf{x}_i$ is assigned to that center $\mathbf{c}_{k^{m-1}(i)}^{m-1}$ minimizing

$$d\left(\mathbf{x}_i, \mathbf{c}_l^{m-1}\right)$$

over choice of l (creating K clusters around the centers) and all of the $\mathbf{c}_k^{m-1}$ are replaced with the corresponding cluster means

$$\mathbf{c}_k^m = \frac{1}{\#\text{ of } i \text{ with } k^{m-1}(i) = k} \sum I\left[k^{m-1}(i) = k\right] \mathbf{x}_i$$

This iteration goes on to convergence.

K-Means Algorithm

Multiple Starts and Comparisons Across K

Multiple random starts for a given K (and then minimum values found for each K) are then compared in terms of

$$\text{Total Dissimilarity}\,(K) = \sum_{k=1}^{K} \sum_{\mathbf{x}_i \text{ in cluster } k} d\left(\mathbf{x}_i, \mathbf{c}_k\right)$$

for $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K$ the final means produced by the iterations. Then considering the monotone sequence of Total Dissimilarities, the object is to identify a value K beyond which there seem to be diminishing returns for increased K to use as a final number of clusters.

Medoid-Based Method

K-Medoid Algorithm First Step

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 "centers" to be original items. This algorithm begins with some set of K distinct "medoids" $\mathbf{c}_1^0, \mathbf{c}_2^0, \ldots, \mathbf{c}_K^0$ that are a random selection from the r items $\mathbf{x}_i$. Then each $\mathbf{x}_i$ is assigned to that medoid $\mathbf{c}^0_{k^0(i)}$ minimizing

$$d\left(\mathbf{x}_i, \mathbf{c}_l^0\right)$$

over choice of l (creating K clusters associated with the medoids). All of the $\mathbf{c}_k^0$ are replaced with $\mathbf{c}_k^1$ the corresponding minimizers over the $\mathbf{x}_{i'}$ belonging to cluster k of the sums

$$\sum_{i \text{ with } k^0(i)=k} d\left(\mathbf{x}_i, \mathbf{x}_{i'}\right)$$

Medoid-Based Method

K-Medoid Algorithm

At stage m with all $\mathbf{c}_k^{m-1}$ available, each $\mathbf{x}_i$ is assigned to that medoid $\mathbf{c}_{k^{m-1}(i)}^{m-1}$ minimizing

$$d\left(\mathbf{x}_i, \mathbf{c}_l^{m-1}\right)$$

over choice of l (creating K clusters around the medoids). All of the $\mathbf{c}_k^{m-1}$ are replaced with $\mathbf{c}_k^m$ the corresponding minimizers over the $\mathbf{x}_{i'}$ belonging to cluster k of the sums

$$\sum_{i \text{ with } k^{m-1}(i)=k} d\left(\mathbf{x}_i, \mathbf{x}_{i'}\right)$$

This iteration goes on to convergence. Multiple random starts for a given K (and then minimum values found for each K) are compared in terms of

$$\sum_{k=1}^{K} \sum_{\mathbf{x}_i \text{ in cluster } k} d\left(\mathbf{x}_i, \mathbf{c}_k\right)$$

for $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K$ the final medoids produced by the iterations.

Hierarchical Methods

Generalities

There are both **agglomerative**/bottom-up methods **and divisive**/ top-down methods of **hierarchical clustering**. To apply a hierarchical method, one must first choose a method of using dissimilarities for items to define dissimilarities for *clusters*. Three common 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, the dissimilarity of C_1 and C_2 might be defined as

1. $D\left(C_1, C_2\right) = \min\left\{d_{ij} \middle| i \in C_1 \text{ and } j \in C_2\right\}$ (this is the "single linkage" or "nearest neighbor" choice),
2. $D\left(C_1, C_2\right) = \max\left\{d_{ij} \middle| i \in C_1 \text{ and } j \in C_2\right\}$ (this is the "complete linkage" choice), or
3. $D\left(C_1, C_2\right) = \frac{1}{\#C_1 \cdot \#C_2} \sum_{i \in C_1,\, j \in C_2} d_{ij}$ (this is the "average linkage" choice).

Hierarchical Methods

An Agglomerative Algorithm

An **agglomerative hierarchical clustering** algorithm then operates as follows. Every item $\mathbf{x}_i, i = 1, 2, \ldots, r$ initially functions as a singleton cluster. The minimum d_{ij} for $i \neq j$ is found and the corresponding two items are placed into a single cluster (of size 2). Then when there are m clusters, the two clusters with minimum dissimilarity are merged 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 **dendogram**, where cutting the dendogram at a given level picks out one of the (increasingly coarse as the level rises) clusterings. Those items clustered together "deep" in the tree/dendogram are interpreted to be potentially "more alike" than ones clustered together only at a high level.

Hierarchical Methods

A Divisive Algorithm

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

$$d\left(\mathbf{x}_i, \mathbf{x}_l\right) < d\left(\mathbf{x}_j, \mathbf{x}_l\right)$$

and to the cluster represented $\mathbf{x}_j$ otherwise. 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 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 dendogram.

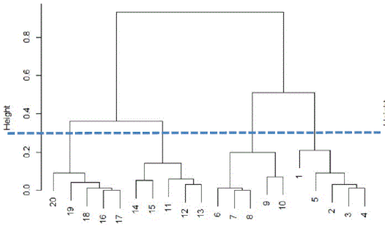
Hierarchical Methods
Thresholding

Both the agglomerative and divisive algorithms may be modified 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 terminates when all pairs of existing clusters have dissimilarities more than t. The divisive version terminates when all dissimilarities for pairs of items in all clusters are below t. Employing a threshold has the potential to shorten the menu of clusterings produced by either of the methods. (Thresholding the agglomerative method cuts off the top of the corresponding full dendogram, and thresholding the divisive method cuts off the bottom of the corresponding full dendogram.)

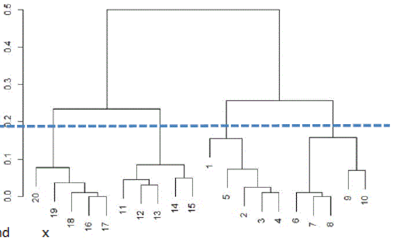
The next slide provides a toy $p = 1$ example. Shown are dendograms for agglomerative clustering for $N = 20$ values for all complete, single, and average linkages. Cuts at 4 clusters make somewhat different groupings. Some experimenting with this data set shows that 4-means clustering places points 1-5, 6-9, 10-15, and 16-20 in clusters.

Hierarchical Methods

Toy Example

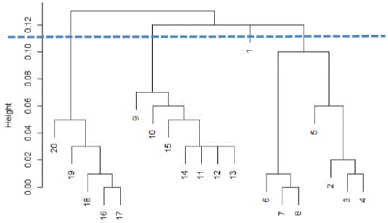


complete

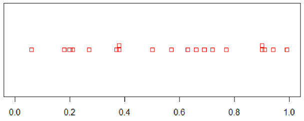


average

ind	x
1	0.06
2	0.18
3	0.20
4	0.21
5	0.27
6	0.37
7	0.38
8	0.38
9	0.50
10	0.57
11	0.63
12	0.66
13	0.69
14	0.72
15	0.77
16	0.90
17	0.90
18	0.91
19	0.94
20	0.99



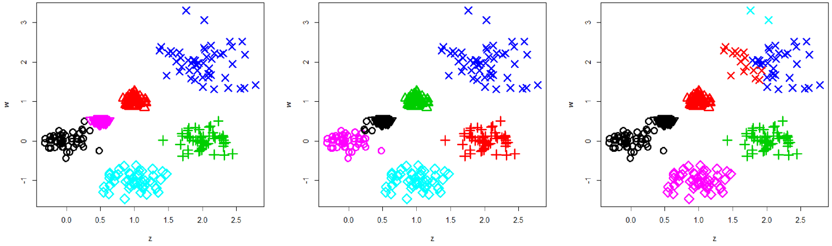
single



Partitioning and Hierarchical Methods

Another Toy Example

Below are plots of $p = 2$ data pairs. The left indicates (by both color and symbol) how the pairs were generated from 6 bivariate normal distributions. The center indicates the result of 6-means clustering. The right shows complete linkage agglomerative clustering cut at 6 clusters. The latter 2 are clearly different.



Model-Based Clustering

Mixture Model

A completely different approach to clustering into K clusters is based on mixture models. For purposes of producing a clustering, one might act as if items $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_r$ are realizations of r iid random variables with parametric marginal density

$$g\left(\mathbf{x} \middle| \boldsymbol{\pi}, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K\right) = \sum_{k=1}^{K} \pi_k f\left(\mathbf{x} \middle| \boldsymbol{\theta}_k\right) \tag{1}$$

for probabilities $\pi_k > 0$ with $\sum_{k=1}^{K} \pi_k = 1$, a fixed parametric density $f\left(\mathbf{x} \middle| \boldsymbol{\theta}\right)$, and parameters $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K$. (Without further restrictions the family of mixture distributions specified by density (1) is not identifiable, but we'll ignore that fact for the moment.)

Model-Based Clustering

K-Class Classification Model With Latent y

A way to think about this formalism is in terms of a K-class classification model where values of y are latent/unobserved/completely fictitious. The model development for classification implies that with $g_k(\mathbf{x}) = f(\mathbf{x}|\boldsymbol{\theta}_k)$ that model produces mixture density (1) as the marginal density of $\mathbf{x}$. Further, in the model including a latent y,

$$P[y = k|\mathbf{x}] = \frac{\pi_k f(\mathbf{x}|\boldsymbol{\theta}_k)}{\sum_{k=1}^{K} \pi_k f(\mathbf{x}|\boldsymbol{\theta}_k)}$$

is the (Bayes posterior) probability that $\mathbf{x}$ was generated by component k of the mixture. It then makes sense to define as "clusters" of observations $\mathbf{x}_i$, **groups that would be similarly classified by an optimal classifier**.

Model-Based Clustering

K-Class Classification Model With Latent y

That is, cluster k can be the set of $\mathbf{x}_i$ for which

$$\frac{\pi_l f\left(\mathbf{x}_i|\boldsymbol{\theta}_l\right)}{\sum_{k=1}^{K} \pi_k f\left(\mathbf{x}_i|\boldsymbol{\theta}_k\right)} \quad \text{or equivalently} \quad \pi_l f\left(\mathbf{x}_i|\boldsymbol{\theta}_l\right)$$

is maximized across l by the index k. In practice, $\boldsymbol{\pi}, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K$ must be estimated and estimates used in place of parameters in defining clusters. An implementable clustering method is to define cluster k (say, C_k) to be the set of $\mathbf{x}_i$ for which k maximizes

$$\widehat{\pi}_l f\left(\mathbf{x}_i|\widehat{\boldsymbol{\theta}}_l\right) \tag{2}$$

across values of l.

The `mclust()` algorithm in `R` when applied to the 300 data pairs used in the example on slide 17 divides cases into exactly the 6 groups indicated on the left panel of that slide. This is, of course, consistent with the fact that the algorithm is built on exactly the kind of MVN models used in the data generation.

Model-Based Clustering
Model Fitting

Given the lack of identifiability in the unrestricted mixture model, it might appear that prescription (2) could be problematic. But it is not. While the likelihood

$$L(\boldsymbol{\pi}, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K) = \prod_{i=1}^{r} g(\mathbf{x}_i | \boldsymbol{\pi}, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K)$$

will have multiple maxima, using any maximizer for an estimate of the parameter vector will produce the same set of clusters (2). It is common to employ the "EM algorithm" in the maximization of $L(\boldsymbol{\pi}, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K)$ (the finding of one of many maximizers). However, strictly speaking, that algorithm is not intrinsic to the basic notion here.