

Statistical Machine Learning-1

Introduction

Stephen Vardeman

Analytics Iowa LLC

January 2018

Statistics for "Big Data" (AKA "Machine Learning" and "Data Analytics")

Notation/Set-up

N cases of p or $p + 1$ variables, x_1, x_2, \dots, x_p and possibly y :

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

- **variables** \leftrightarrow **features**
- **cases** \leftrightarrow **instances**
- $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{ip})'$ case i vector of x values.

Statistics for Big Data

Realm of Application, Objective, New Possibility

- Big N (and potentially big p) and little fundamental interest in model parameters (or exactly how much is known about them)
- (As in all of statistics) Identifying, (?describing?,) and enabling the practical use of simple (low-dimensional/low-order) structure in the $N \times p$ or $N \times (p + 1)$ data array
- The potential to choose among a broad spectrum of method "complexities" to match one to a given application

Machine Learning/Statistics for Big Data

Types of Problems

- **supervised learning** problems¹, where there is a response/output variable or **target**, y , and the problem is finding a function of p inputs, $f(\mathbf{x})$, that approximates y
 - Where y is measured/continuous, the problem is typically called **prediction**
 - Where y takes values in a finite set $\{1, 2, \dots, K\}$, the problem is **classification** or **pattern recognition**
- **unsupervised learning** problems, where there is no response variable and the objective is to identify relationships between the p variables x or commonalities in segments of the N cases or p variables
 - Standard versions are **clustering**, **principal components analysis**, and **multi-dimensional scaling**

¹Here the input variables \mathbf{x} are sometimes called **covariates** and the $N \times (p + 1)$ data array **T** comprises the **training data**

What "New" Issues Arise in Statistics for "Big Data"?

Computational Limitations (Not Really Addressed Here)

Where N and/or p is large, computational limitations can make straightforward implementation of standard methods impractical or even impossible

- Sometimes clever implementations (for example, employing parallelization or specialized hardware) make application of standard methods feasible
- Other times, new methods must be developed

New Issues With "Big Data"

The Possibility of Profitably Fitting Flexible Forms of Predictors

- For N big and p small, standard statistical prediction methods (like multiple linear regression) produce *precisely fit* but *relatively crude* predictors
 - Forms are only "first approximations" to a real relationship between predictors \mathbf{x} and output y and fail to really make full use of the available information
- There is the possibility of increasing " p " by (implicitly or explicitly) building additional features from existing ones and/or simply using more sophisticated and flexible forms for prediction
- But there is also the potential to "over-do" and make p too large or the method too flexible

One must somehow match predictor complexity to the real information content of a training set

New Issues With "Big Data"

The Curse of Dimensionality

- If p is big, \mathcal{R}^p is "huge"
 - Intuition about how many cases are required to "fill up" even an intuitively small part of p -space is poor
 - Essentially *any* data set with large p is necessarily "sparse"
- Further, the potential complexity of functions of p variables explodes exponentially in p
- When p is large, it is essentially guaranteed that if one uses a method that is "too" flexible in terms of the relationships between variables it permits, one will be found, *real/fundamental/reproducible or not*

These issues are aspects of **"the curse of dimensionality"**

New Issues With "Big Data"

"Over-Fitting" in Supervised Learning

- The (common for large p) possibility that a data set is (sparse and) not really adequate to support the use of a flexible supervised statistical learning method can easily lead to **over-fitting**:

too closely following an apparent pattern in a (sparse) training set, that then generalizes/extrapolates poorly to cases outside the training set

- This is equally as unattractive as failing to follow a clear well-established pattern because one's prediction methodology is too simple/inflexible

What to Do About Overfitting?

Explicit Optimization of Predicted Performance as a Function of Method Complexity

The standard way of choosing among "big data" statistical procedures is:

- To define both a reliable measure of estimated/predicted performance (like an estimated prediction mean square error) and a measure of complexity for a predictor
- Then one attempts to optimize (by choice of complexity) the predicted performance²
- Performance prediction almost always employs some form of **"holdout" sample** (whereby performance is predicted using *data not employed in fitting*)

²This approach balances risks of overfitting and "model bias" (where a fitted form is not adequate to usefully represent the real relationship between x and y)

Up-Front Work in Predictive Analytics—"Data Mining"

Development of Plausible "Feature" Vectors

- Reduction of all information available and potentially relevant to predicting y to values of p input variables³ (that encode relevant "features" of the N cases) is an essential and highly critical activity⁴
- Only if one defines good features/variables (parsimoniously representing the N cases in ways compatible with the prediction methodologies considered) does sound statistical methodology have a chance of being practically helpful

In this way, the hard work begins substantially *before* the formal technical subjects addressed in these modules come into play and typically continues even after initial attempts at prediction

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

⁴This is particularly true where many disparate sources are used to create the training set, \mathbf{T} , available for analysis

Up-Front Work—Centering and Scaling Variables

Units and Scales

- x (and y) variables are typically in different units and often represent conceptually different quantities (e.g., voltage, temperature, and distance)
 - In some analyses this causes no logical problems
 - In others (particularly ones based on inner products of data vectors or distances between them and/or where "sizes" of model coefficients are important) one gets fundamentally different results depending upon the scales used
- One surely doesn't want predictions to depend upon whether a distance is expressed in km or in nm
- And the whole notion of the \mathfrak{R}^2 "distance between two data vectors" where different units are involved is problematic (e.g., what is $\sqrt{(3 \text{ kV})^2 + (2^\circ \text{ K})^2}$ supposed to mean?)

Centering and Scaling Variables

Standardization of Predictors and Centering of Response

- One approach to eliminating logical difficulties that arise in using methods where scaling/units of variables matters, is to standardize predictors x (and center any quantitative response variable, y) before beginning analysis
 - If a raw feature x has (in \mathbf{T}) a sample standard deviation⁵ s_x and a sample mean \bar{x} , one replaces it with a feature

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

(thereby making all features unit-less)

- Conclusions about standardized input x' and centered response $y' = y - \bar{y}$ then translate naturally to conclusions about raw variables via

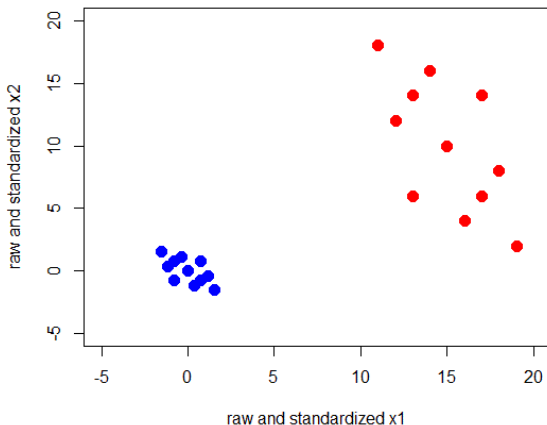
$$x = s_x \cdot x' + \bar{x} \quad \text{and} \quad y = y' + \bar{y}$$

⁵The " N " divisor in place of the " $N - 1$ " divisor here is slightly simpler, as it makes the x columns of \mathbf{T} have \Re^N norm \sqrt{N} (as opposed to $\sqrt{N-1}$)

Centering and Scaling Variables

Standardization of Predictors

Below is a plot of a small fake $p = 2$ raw data set (red) and corresponding standardized data set (blue). (Raw data means and standard deviations are $\bar{x}_1 = 15$, $\bar{x}_2 = 10$, $s_1 = 2.61$, and $s_2 = 5.22$.)



Statistical Machine Learning-2B

Using Cross-Validation to Choose Between Predictors

Stephen Vardeman

Analytics Iowa LLC

January 2020

Choosing a Level of Complexity

The "One-Standard Error of the Cross-Validation Error" Rule

A common method of choosing between levels of complexity has been this. For the complexity producing the smallest realized cross-validation error, one computes a standard error for the prediction error. That is, for each "fold" \mathbf{T}_k , one computes a k th "test error" for \hat{f}^k obtained by fitting on $\mathbf{T} - \mathbf{T}_k$ and evaluating on \mathbf{T}_k

$$CV_k(\hat{f}) = \frac{1}{\text{number of cases in } \mathbf{T}_k} \sum_{\text{cases in } \mathbf{T}_k} L(\hat{f}^k(\mathbf{x}_i), y_i)$$

Then for SD_K the sample standard deviation of these $CV_1(\hat{f})$, $CV_2(\hat{f})$, \dots , $CV_K(\hat{f})$, the standard error of interest is SD_K / \sqrt{K} . One then selects for use the least complex predictor with its own corresponding cross-validation error no larger than

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

Choosing a Level of Complexity

The "Minimum (Average) Cross-Validation Error" Rule

The most obvious and most aggressive way of using $CV(\hat{f})$ (or better $\overline{CV(\hat{f})}$) to choose a predictor is to simply use the \hat{f} minimizing the function $CV(\cdot)$ (or $\overline{CV(\cdot)}$).

It is an important and somewhat subtle point that if

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

$CV(\tilde{f})$ is **not** a valid cross-validation error for a predictor that arises from "picking a winner" on the basis of $CV(\cdot)$ (or $\overline{CV(\cdot)}$).¹ The issue is that while $CV(\hat{f})$ (or $\overline{CV(\hat{f})}$) *can* legitimately guide the choice of \hat{f} , its use is then actually part of a larger program of "predictor development" than that represented by any single argument of $CV(\cdot)$ (or $\overline{CV(\cdot)}$).

¹Intuition suggests that it will typically be optimistic as representing Err for the pick-the-winner predictor.

Predicting Performance of a "Pick the Winner" Predictor

Cross-Validation for "Pick the CV-Winner"

In order to assess the likely performance of \tilde{f} , via cross-validation, **inside each remainder $\mathbf{T} - \mathbf{T}_k$** one must

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

then predict on \mathbf{T}_k using \tilde{f}^k . It is the values $\tilde{f}^{k(i)}(\mathbf{x}_i)$ that are used to make a cross-validation error for a predictor derived from optimizing a cross-validation error across a set of predictors.

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

Statistical Machine Learning-2

Supervised Learning Generalities: Decision Theory, Model Flexibility and Fitting, Cross-Validation

Stephen Vardeman

Analytics Iowa LLC

January 2020

Decision Theory and Supervised Learning

Optimal Predictors and Classifiers

In the context of choosing $f(\mathbf{x})$ to track y , suppose that P is a $((p + 1)$ -dimensional) distribution for (\mathbf{x}', y) and $L(\hat{y}, y) \geq 0$ is a (loss) function for penalizing prediction/classification \hat{y} when y holds. For "E" expectation under P , one might hope to minimize expected prediction loss

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

In theory (given P) this is "easy." One chooses $\hat{y} = f(\mathbf{x})$ to minimize

$$E[L(\hat{y}, y) | \mathbf{x}]$$

$f(\mathbf{x})$ is the prediction that minimizes *conditional* (on \mathbf{x}) expected (over y) prediction loss.

Decision Theory and Supervised Learning

Prediction of a Quantitative Output

For squared error loss (SEL)

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

an optimal predictor is

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

the *conditional (on \mathbf{x}) mean output*. This is the usual "regression" context, where one attempts to describe average response as a function of the input vector.

Decision Theory and Supervised Learning

Classification

In (K -class) classification problems where y takes values in $\{0, 1, \dots, K - 1\}$ or $\{1, \dots, K\}$ a standard loss is so-called 0-1 loss¹

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

For this loss, an optimal classifier is $\hat{y} = f(\mathbf{x})$ minimizing

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

or, equivalently the maximizing \hat{y} for

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

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

¹The "indicator" notation, $I[\text{statement}]$, stands for a function that is 1 when "statement" is true and 0 when it is false.

Decision Theory and Supervised Learning

Classification Continued-Insights for $K=2$

For $K = 2$, suppose y takes values in $\{0, 1\}$ and abbreviate $P[y = 1]$ as π (so that $P[y = 0] = 1 - \pi$) and write $g(\mathbf{x}|1)$ and $g(\mathbf{x}|0)$ for the class-conditional densities for \mathbf{x} . Then

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

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

An optimal classifier is

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

This decides in favor of $y = 1$ when $P[y = 1|\mathbf{x}]$ (or equivalently the "likelihood ratio" $g(\mathbf{x}|1) / g(\mathbf{x}|0)$) is large.

Test/Prediction/Generalization Error

Definition of Err

One cannot simply use an optimal predictor f , as one does not have P (the joint distribution of (\mathbf{x}', y)). Instead, one has a training set \mathbf{T} providing information about P and the form, \hat{f} , of a practical predictor can depend upon \mathbf{T} .

Suppose that (for \mathbf{x}_i the case i row vector of feature values) the training data (\mathbf{x}_i, y_i) for $i = 1, \dots, N$ are iid according to P , independent of a new test case $(\mathbf{x}', y) \sim P$. Let $E^{\mathbf{T}}$ be expected value averaging out the random training set and $E^{(\mathbf{x}, y)}$ be expected value averaging out the test case. A figure of merit for \hat{f} is the "prediction"/"generalization"/"test" error

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

Test/Prediction/Generalization Error

Understanding Err

Err is the average loss suffered using predictor \hat{f} . The averaging is done across (hypothetical) selections of training set of size N and (hypothetical) test case.

For SEL prediction it is

$$E^T E^{(x,y)} (y - \hat{f}(x))^2$$

a mean squared prediction error.

For 0-1 loss classification it is

$$E^T E^{(x,y)} I [y \neq \hat{f}(x)]$$

an overall classification error rate.

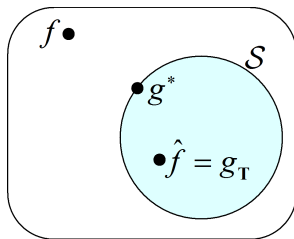
A General Decomposition of Err

Optimal, Restricted Optimal, and Fitted Predictors

A helpful decomposition of Err is available. If $f(\mathbf{x})$ is the optimal predictor of y , \mathbf{T} is used to select a function (say $g_{\mathbf{T}}$) from a class of functions $\mathcal{S} = \{g\}$ having expected losses $E^{(\mathbf{x}, y)} L(g(\mathbf{x}), y) < \infty$, and ultimately one uses as a predictor

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

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



A General Decomposition of Err

Optimal, Restricted Optimal, and Fitted Predictors

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

It follows that $\text{Err} = \mathbb{E}^{\mathbf{T}} \mathbb{E}^{(\mathbf{x}, y)} L(\hat{f}(\mathbf{x}), y) = \mathbb{E}^{\mathbf{T}} \mathbb{E}^{(\mathbf{x}, y)} L(g_{\mathbf{T}}(\mathbf{x}), y)$ can be decomposed into three non-negative terms

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

A General Decomposition of Err

Modeling and Fitting Penalties

These three terms are

- minimum possible error = expected loss of the optimal f
- modeling penalty = the difference between the expected loss of f and that of g^*
- fitting penalty = the difference between the expected loss of $\hat{f} = g_{\mathcal{T}}$ and that of g^*

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

SEL, Err, and "the Variance-Bias Trade-Off"

The Decomposition of Err and SEL

SEL prediction is a particularly important special case of the foregoing where further insight is available. The terminology "variance-bias trade-off" is common because²

- minimum expected loss possible = average (across \mathbf{x}) response variance
- modeling penalty = average (across \mathbf{x}) squared model bias
- fitting penalty = $\left(\begin{array}{c} \text{average (across } \mathbf{x} \text{)} \\ \text{squared fitting bias} \end{array} \right) + \left(\begin{array}{c} \text{average (across } \mathbf{x} \text{)} \\ \text{prediction variance} \end{array} \right)$

²See Vardeman's notes online for details

The Role of Feature Selection

Modeling and Fitting Penalties and the "Richness" of a Feature Set

That both model flexibility (encoded in the size of \mathcal{S}) and fitting impact predictor performance is related to the fact that effective selection of p features is critical. The "richness" of a set of features representing information available for prediction limits the potential effectiveness of prediction, inadequate richness producing big model bias. But if richness is bought at the expense of very large p , then one has a "needle in a haystack" when looking for a good predictor, and poor fitting properties often ensue.³

³If one were infinitely wise, one would pick a single feature that was a perfect predictor of y . If one were completely inept, none of many features one invented would be of any help in prediction. Real analysts are typically neither infinitely wise nor completely inept, and good up-front definition of sensible features together with use of the tools of these slides results in effective real world prediction.

Predicting Predictor Performance in Supervised Learning

In-Sample or Training Error

To choose between predictors or classifiers, one might hope to estimate Err for each and choose one minimizing the estimated error. The most obvious way to try to do this is using the "**training error**"

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

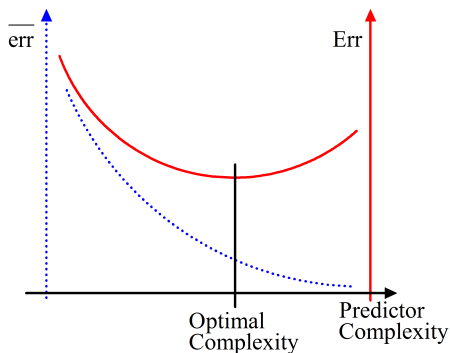
In SEL problems this is an empirical mean squared error and in 0-1 loss classification it is an empirical/training set classification error rate.

But $\overline{\text{err}}$ is not a good estimator of Err . It typically decreases monotonically with increased complexity (without increasing for large complexity), and fails to reliably indicate performance outside the training set.

Performance and Complexity in Supervised Learning

Training Error and Err

The cartoon below illustrates the problem faced in choosing a predictor. $\overline{\text{err}}$ decreases with increased complexity ("low bias" in SEL problems) while Err decreases and then increases. One wants a predictor with approximately optimal complexity (e.g. in light of the "variance-bias trade-off" in SEL problems) and $\overline{\text{err}}$ provides no direction.



Predicting Predictor Performance

Methods

Existing options for reliably evaluating likely predictor performance (and guiding choice of complexity) are:

1. Employing other (besides $\overline{\text{err}}$) training-data based indicators of likely predictor performance, like Mallows' C_p , "AIC," and "BIC."
2. In genuinely large N contexts, holding back some random sample of the training data to serve as a "test set," fit to produce \hat{f} on the remainder, and using

$$\frac{1}{\text{size of the test set}} \sum_{\substack{i \in \text{the} \\ \text{test set}}} L(\hat{f}(\mathbf{x}_i), y_i)$$

to assess likely predictor performance.

3. Employing sample re-use methods like **cross-validation** or the bootstrap to estimate Err .

Predicting Predictor Performance

Cross-Validation

K -fold cross-validation consists of:

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

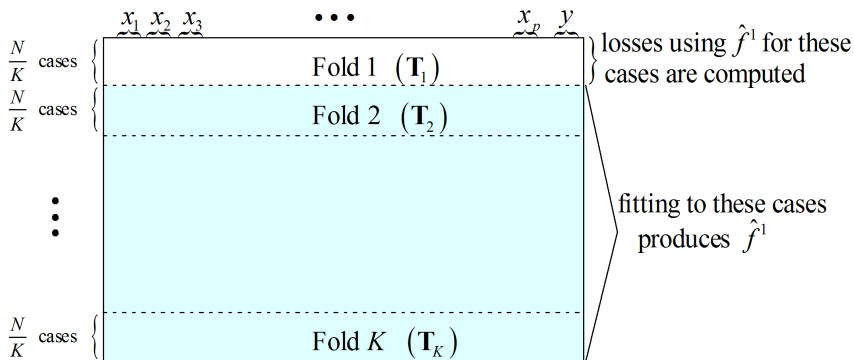
$$CV(\hat{f}) = \frac{1}{N} \sum_{i=1}^N L(\hat{f}^{k(i)}(\mathbf{x}_i), y_i)$$

One hopes that $CV(\hat{f})$ approximates Err . Where computationally feasible, averaging results from **repeated cross-validations** mitigates the arbitrariness of the single random partition of \mathbf{T} .

Predicting Predictor Performance

Cross-Validation


Below is a graphic suggesting roughly how (after putting the N cases into a random order) one breaks \mathbf{T} into folds and computes the part of sum defining $CV(\hat{f})$ for cases in the first fold. (Of course, slightly different pictures are needed for the sums from the other $K - 1$ folds.)



Predicting Predictor Performance

Choice of K for Cross-Validation

$K = N$ is called "leave one out (LOO)" cross-validation. In this case there are sometimes slick computational ways of evaluating $CV(\hat{f})$. Cross-validation actually estimates Err for a training set of size approximately $N(1 - K^{-1})$, so there is potential bias⁴ that typically decreases with increasing K . The statistical folklore has been that LOO cross-validation also tends to have large variance as an estimator of Err and so consideration of a bias-variance trade-off for representation of Err by $CV(\hat{f})$ has made $K = 5$ and $K = 10$ standard choices in practice. But recent work by Zou and Wang has called this folklore into question and points strongly to the choice of $K = N$ (LOOCV) where it is computationally feasible.

⁴For the use to which the cross-validation error is put, bias is a problem only if it is *not constant* across choices of predictors and their complexities. 

Predicting Predictor Performance

A Caution: COMPLETE/SEPARATE Calculation is Required for EACH Remainder

It is important to emphasize that *all of* whatever will be done with a training set to produce \hat{f} must be done *separately* (thus K times) *in each* of the remainders $\mathbf{T} - \mathbf{T}_k$, to produce the predictors \hat{f}^k . **It does not suffice** to somehow use the entire data set once to cover a first step in predictor development and then subsequently operate separately in the remainders. IN PARTICULAR, any "preprocessing" of training data \mathbf{T} that will be done to make \hat{f} must be redone for every $\mathbf{T} - \mathbf{T}_k$, to produce the \hat{f}^k .

As a very simple illustration, if columns of a matrix of predictor values are to be standardized before fitting \hat{f} , columns of the smaller matrices of predictor values in $\mathbf{T} - \mathbf{T}_k$ must be *separately* standardized before fitting the \hat{f}^k . One cannot simply use part of standardized columns of predictors made from the entire training set.

Predicting Predictor Performance

Again: COMPLETE/SEPARATE Calculation is Required for EACH Remainder

This is a big deal. The effect of failing to redo all calculations that depend upon values in the training set separately in every remainder is to ultimately use the entire training set in making each \hat{f}^k . This is exactly what cross-validation is intended to avoid! Almost always when this issue is ignored, a supposed cross-validation error is too optimistic.

The practical pressure to "cheat" here is strong, particularly where complicated "custom" data preprocessing is involved. But if the preprocessing is data-dependent (exactly what is done depends upon the numbers in all cases represented in \mathbf{T}) there is no avoiding the issue. A little experience will teach the hard lesson that if it is ignored the resulting " $CV(\hat{f})$ " values are completely unreliable as guides in predictor selection.

Statistical Machine Learning-3

More Simple SEL Prediction Generalities and Nearest Neighbor Predictors

Stephen Vardeman

Analytics Iowa LLC

January 2018

Mean to SLR to MRL to ... k -NN

A Spectrum of Possible Predictors

To introduce thought processes of modern predictive analytics and give a first look at a highly flexible predictor, consider SEL prediction of a quantitative y based on a p -dimensional input $\mathbf{x} = (x_1, x_2, \dots, x_p)$. Many predictors are possible, including (roughly in order of increasing "complexity")

1. $\hat{y} = \bar{y}$
2. simple linear regression prediction based on a single x_j
3. multiple linear regression prediction based on some x_j s
4. \vdots
5. k -nearest neighbor predictors.

1 through 3 above should be familiar. 4 will be filled-in in subsequent sessions. " k -nearest neighbor" is possibly the most flexible SEL prediction method available.

k-NN Predictors

Local Averaging of Training Set Responses

The idea of nearest neighbor prediction is to approximate the theoretical/long-run mean y at input vector \mathbf{x} , using a training-set-mean "near" \mathbf{x} (typically there will be few or no cases with input vector *exactly* \mathbf{x} , and hence the "near"). For "nearness" to be unit-free/meaningful we'll assume that inputs have been standardized. The k -neighborhood of \mathbf{x} is the set of k training cases with \mathbf{x}_i closest to \mathbf{x} in \mathfrak{R}^p and the corresponding predictor is

$$\hat{f}^{k\text{NN}}(\mathbf{x}) = \frac{1}{k} \sum_{i \text{ s.t. } \mathbf{x}_i \text{ is in the neighborhood}} y_i$$

Two hypothetical nearest neighbor predictions for a $p = 2$ case are illustrated on the next slide.

k-NN Predictions

p=2 Toy Example

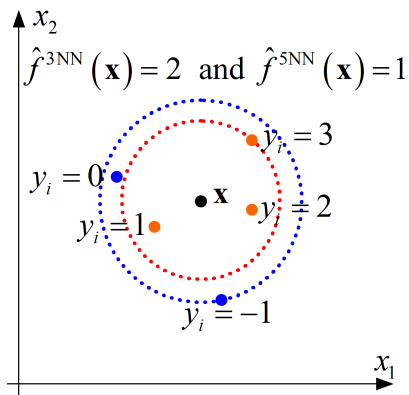


Figure: 5 Nearest Neighbors of \mathbf{x} and $\hat{f}^{3\text{NN}}$ and $\hat{f}^{5\text{NN}}$

Comparing Predictors on a Real Case

Ames Housing Data Set

Sales prices and values of 14 predictors for 88 homes sold in Ames, Iowa 2002-2003 are in a data set on the page:

<http://www.public.iastate.edu/~vardeman/stat342/stat342.html>

The R package `caret` can be used to do repeated cross-validation for ordinary regression, k -NN fitting (this latter with variables correctly newly standardized for each fold), and many other prediction methods as well. Some results comparing predicted performance for the sample mean, 3 different simple linear regressions, the full multiple linear regression, and several k -NN fits are on the next slide. Among the linear models, the constant is less complex than any single SLR, each of which is less complex than the MLR. k -NN fits decrease in complexity as k increases.

Comparing Predictors on a Real Case

Fits to the Ames Housing Data

Predictor	8-Fold CV RMSE	LOOCV RMSE	Predictor	8-Fold CV RMSE	LOOCV RMSE
5NN	22181	23410	\bar{y}	36491	37567
6NN	21799	23448	SLR(land)	33577	34495
7NN	21245	22177	SLR(fireplace)	28668	28847
8NN	21540	22409	SLR(size)	27608	28452
9NN	21537	22886	MLR(all)	22335	22600

Among the predictors considered here, the 7NN predictor appears to have the most appropriate level of complexity and the corresponding smallest cross-validation errors. It seems better than the less complex 8 and 9NN predictors and the more complex 5 and 6NN predictors. The simple mean and 3 SLR predictors are too simple and even the overall MLR is not as good as the 7NN predictor.

NN Prediction

Large p Limitations and Value

Nearest neighbor prediction does extremely well in the Ames Housing Price problem despite the fact that $N = 88$ is not so large relative to $p = 14$. Often its usefulness is restricted to very large N and small p problems. (The curse of dimensionality means that in a complex large p problem, "neighbors" are rarely "close" and can thus often fail to well-represent each other.)

NN predictors

1. are easy to understand and point in the direction of what one needs to do to make good predictions,
2. provide a kind of "most flexible possible" predictor, and
3. together with various linear regressions provide a first look at **cross-validation** and **the importance of matching predictor methodology to the real information content of a dataset.**

Statistical Machine Learning-4

Making Usable Flexible Predictors (Mostly) in 1-D (Basis Functions and Smoothing)

Stephen Vardeman

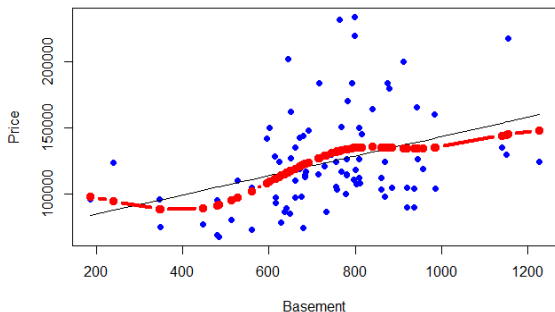
Analytics Iowa LLC

January 2018

What if y Isn't Linear in x?

"Improving on" a Raw Variable, x

Below is a plot of Price versus Basement square feet for the Ames house data (blue dots). Price is only very approximately linearly related to Basement size. Red dots represent values of a smooth transform (**whose development is enabled by "large N" relative to 1-D**) of basement area that might be a better predictor than area itself.



More Flexible Forms From x

Standard "Transformations"

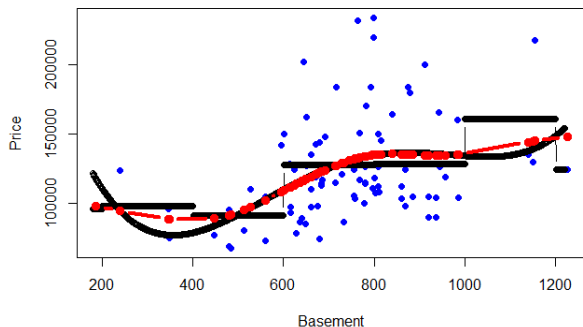
"Transformations" of a 1-D predictor x are standard in regression analysis.
Linear combinations of functions

- $1, x, x^2, x^3, \dots$ are used to make polynomials in x
- $1, \sin 2\pi x, \cos 2\pi x, \sin 4\pi x, \cos 4\pi x, \dots$ are used to fit periodic relationships between x and y
- $I[x \geq c_1], I[x \geq c_2], I[x \geq c_3], I[x \geq c_4], \dots$ for constants $c_1 < c_2 < c_3 < \dots$ can be used to fit (step-function) relationships between x and y constant on intervals $[c_j, c_{j+1})$

More Flexible Forms From x

Transforming "Basement"

5th degree polynomial and step function predictions (for $c_1 = 200, \dots, c_6 = 1200$) are in black below. The former are unpleasant because of poor extrapolation properties and the latter because of discontinuities and lack of good ways of choosing the number and locations of the c_i .

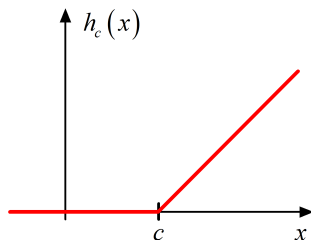


More Flexible Predictors From x

Other "Basis" Functions

For "big data" purposes there are better sets of "**basis functions**" (transformations) for linear fitting than those on panel 3. And beyond their *direct* use, they arise *indirectly* in "smoothing spline" technology (that produced the "red" predictions). Because they are related to **flexible and smooth** prediction, we next briefly consider them.

$h_c(x) = (x - c) I[x > c]$ is a "hinge function" located at c . It is (as below) 0 to the left of c and increasing with slope 1 from $x = c$.



More Flexible Predictors From x

Piecewise Linear Regression Splines

For some set of "knots" $c_1 < \dots < c_k$ one form of (**piece-wise linear regression spline predictor**) is a linear combination of hinge functions and the functions 1 and x . These are of the form

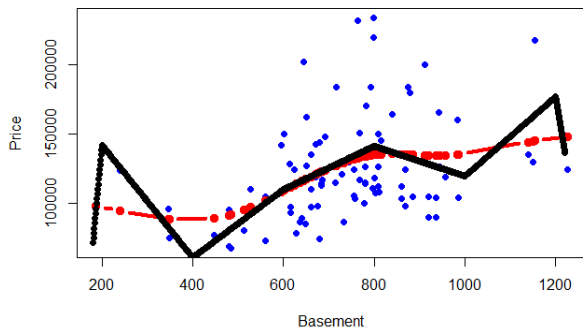
$$\alpha + \beta_0 x + \sum_{j=1}^k \beta_j h_{c_j}(x)$$

(that for fixed c_j s can be fit using a MLR program). A fitted version of this kind of predictor for Price and Basement (for $c_1 = 200, \dots, c_6 = 1200$) is plotted on the next slide. The predictor is continuous and linear between knots. It has esthetic deficiencies in terms of lack of smoothness and extrapolation properties.

More Flexible Predictors From x

One Piecewise Linear Regression Spline for the Housing Data

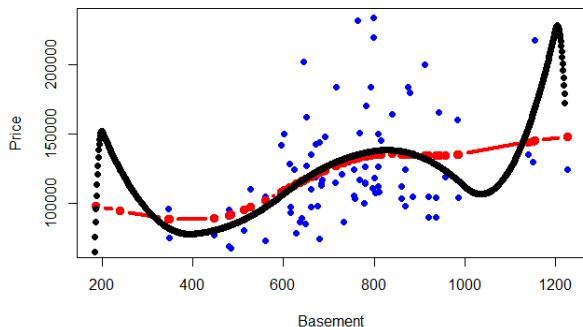
The "corners" on the plot are sharp changes in derivative at the knots. They can be smoothed out by changing the set of basis functions to $1, x, x^2$, and a series of *squared* hinge functions $h_{c_j}^2(x)$.



More Flexible Predictors From x

One Piecewise Quadratic Regression Spline for the Housing Data

For the same set of knots, this set of 9 basis functions produces the fitted **piecewise quadratic regression spline predictor** below. This is smoother, but little more attractive than the previous (piecewise linear) predictor.



More Flexible Predictors From x

Sources of Bad Behavior in the Example—And 1-D Smoothing

The unappealing appearance of the piecewise polynomial regression spline predictor comes from 3 interacting sources

- The locations of the two extreme knots
- The fact that the predictor is quadratic to the left of the 1st knot and to the right of the last one
- The fact that the fitting was done by least squares

We proceed to a form of modern prediction that behaves far better than piecewise polynomial regression splines with arbitrarily chosen knots. This is the technology of **smoothing splines**.

1-D Smoothing Splines

A Function Optimization Problem

For a **(complexity) penalty weight** $\lambda > 0$, consider (for $a \leq \min \{x_i\}$ and $\max \{x_i\} \leq b$) an \hat{f}_λ minimizing (over choices of functions g on $[a, b]$ with second derivative g'')

$$\sum_{i=1}^N (y_i - g(x_i))^2 + \lambda \int_a^b (g''(x))^2 dx$$

Such an \hat{f}_λ must be a **piecewise cubic regression spline constrained to be linear outside** $[\min \{x_i\}, \max \{x_i\}]$ **with knots at distinct training set values** x_i . For $\{b_1, b_2, \dots, b_N\}$ a (data-dependent) set of basis functions for such splines (assuming that the N values x_i are all different)

$$\hat{f}_\lambda(x) = \sum_{j=1}^N \hat{\beta}_{\lambda j} b_j(x)$$

where the $\hat{\beta}_{\lambda j}$ depend upon λ .

1-D Smoothing Splines

Basis Functions and Related Matrices

The b_1, b_2, \dots, b_N can be linear combinations of the $N + 4$ functions

$$1, x, x^2, x^3, h_{x_1}^3(x), h_{x_2}^3(x), \dots, h_{x_N}^3(x)$$

where the linear combinations are chosen to enforce the "linear outside $[\min \{x_i\}, \max \{x_i\}]$ " condition.

Further, with matrices

- $\mathbf{H}_{N \times N} = (b_j(x_i))$ giving data-dependent transforms of the x_i
- $\mathbf{\Omega}_{N \times N} = \left(\int_a^b b_j''(t) b_l''(t) dt \right)$ collecting integrals of products of second derivatives of the basis functions

the vector of coefficients $\beta_\lambda \in \mathfrak{R}^N$ minimizes *quadratic function*

$$Q(\beta) = (\mathbf{Y} - \mathbf{H}\beta)' (\mathbf{Y} - \mathbf{H}\beta) + \lambda \beta' \mathbf{\Omega} \beta$$

1-D Smoothing Splines

Fitting and Predictor Complexity

This is not ordinary least squares, but rather *penalized* (by the quadratic form $\beta' \Omega \beta \geq 0$) *least squares fitting!* Without the penalty, this would simply be MLR with matrix of predictors \mathbf{H} . But as it is,

$$\hat{\beta}_\lambda = (\mathbf{H}'\mathbf{H} + \lambda\Omega)^{-1} \mathbf{H}'\mathbf{Y}$$

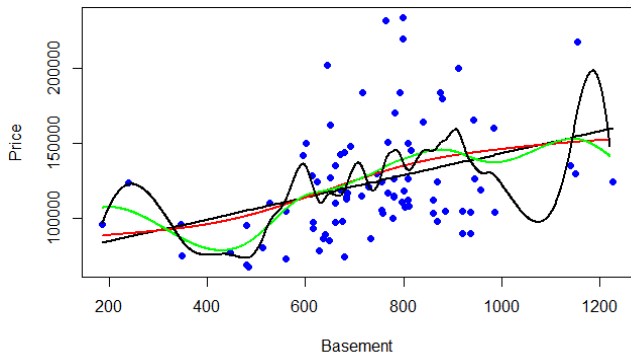
and as λ goes from 0 to ∞ the (smoothing spline) predictor goes from MLR (with matrix of predictors \mathbf{H}) *that interpolates the training data* to simple linear regression on x .

The wide variety of smoothing splines for the Price versus Basement data on the next panel illustrates the effect of λ on the nature of the predictors. (Plots for $\lambda = \infty, 20000000, 200000$, and 2000 are shown.)

1-D Smoothing Splines

Splines for the Ames Housing Data and Complexity

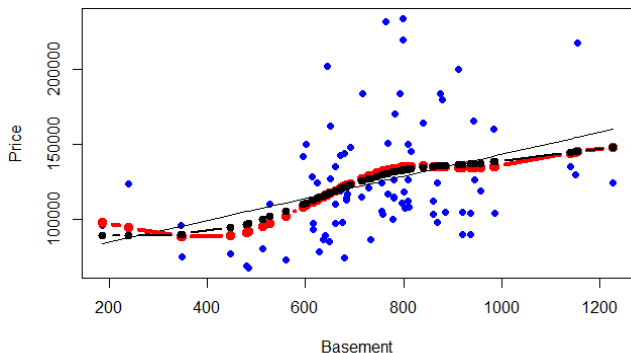
λ is clearly a complexity parameter ... and can be chosen by cross-validation.



1-D Smoothing Splines

Cross-Validation and Choice of Smoothing Spline

Red dots on earlier plots correspond to $\lambda \approx 10600000$. LOOCV suggests the slightly more linear spline for $\lambda \approx 20000000$. Below, data are in blue, red points are as before, and black points are for the LOOCV choice.



Other 1-D Smoothing Methodology

Locally Weighted Means and Polynomials

A popular and effective alternative to 1-D smoothing splines is "weighted local regression." Because smoothing splines already begin to indicate what is possible and make somewhat better connections to other methods of prediction to be discussed here, we will not present this alternative. But spline and local regression technologies are somewhat interchangeable and often give similar results (at least when values of x are more or less uniformly distributed).

Smoothing for $p > 1$

Direct Creation of Smooth Flexible p -Dimensional Predictors

There is a 2-dimensional analogue of 1-D smoothing splines that is known as "thin plate splines" (that involves penalized fitting of a specific kind of "radial basis function network"). And there is a natural p -dimensional version of locally weighted regression smoothing.

But direct application of smoothing methods generally becomes more and more problematic as p grows. The curse of dimensionality sets in, the \mathbf{x}_i in a training set are of necessity sparse in \mathbb{R}^p (unless they themselves are highly internally structured) and direct smoothing works less and less effectively. Something else must be done to create effective flexible p -dimensional predictors for large p .

Statistical Machine Learning-5

Building Flexible Large p Predictors: MARS and GAM

Stephen Vardeman

Analytics Iowa LLC

February 2020

Building Flexible Predictors When p is Large

Hints from Smoothing

Unless p is very small, direct application of smoothing to the entirety of \mathbf{x} is typically not effective. But some of the ideas met in low-dimensional smoothing *can* be part of practical solutions to the problem of developing effective predictors without suffering from over-fitting.

We next consider two paths to practical development of flexible predictors for large p and provide brief introductions to multivariate adaptive regression splines (**MARSs**) and to generalized additive models (**GAMs**).

Multivariate Adaptive Regression Splines (MARS)

Hinge/"Hockey-Stick" Functions and Their Products

Motivated by 1-D regression- and smoothing- spline developments, MARS is a high-dimensional forward-selection-regression spline-like methodology based on "hockey-stick" functions and their products as data-dependent basis functions in p -D.

MARS uses *data-dependent* basis functions built on (1 and) the Np pairs of functions of $\mathbf{x} \in \mathfrak{R}^p$ (based on 1-D hinge functions)¹

$$g_{ij+}(\mathbf{x}) = (x_j - x_{ij})_+ = h_{x_{ij}}(x_j) \quad \text{and} \\ g_{ij-}(\mathbf{x}) = (x_{ij} - x_j)_+ = (x_{ij} - x_j) + h_{x_{ij}}(x_j)$$

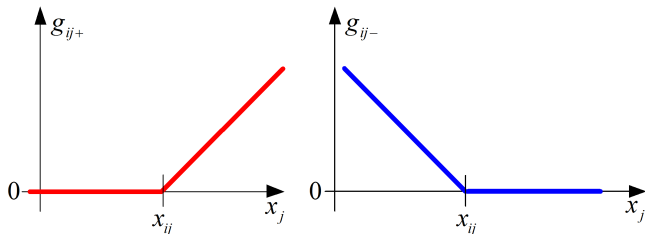
MARS builds predictors sequentially, making use of these "reflected pairs" (portrayed on the next slide).

¹ x_{ij} is the j th coordinate of the i th input training vector and both $g_{ij+}(\mathbf{x})$ and $g_{ij-}(\mathbf{x})$ depend on \mathbf{x} **only through the j th coordinate of \mathbf{x}** . The notation $(z)_+$ is short for $z \cdot I[z > 0]$.

MARS

Reflected Pairs and Model Building

Below are representations of how the functions (of $\mathbf{x} \in \mathbb{R}^p$) $g_{ij+}(\mathbf{x})$ and $g_{ij-}(\mathbf{x})$ depend upon x_j .



Exactly how to build up (in a forward selection fashion) a linear combination these functions and various products of them is the "special sauce" of any particular MARS implementation. (There are variable selection, variable deletion, and stopping rules to be chosen.)

Some versions of MARS begin by identifying a reflected pair $g_{ij+}(\mathbf{x})$ and $g_{ij-}(\mathbf{x})$ so fitting by ordinary least squares

$$\hat{\beta}_0 + \hat{\beta}_{11}g_{ij+}(\mathbf{x}) + \hat{\beta}_{12}g_{ij-}(\mathbf{x})$$

has the best *SSE* possible. Call the selected functions

$$g_{11} = g_{ij+} \text{ and } g_{12} = g_{ij-}$$

and set

$$\hat{f}_1(\mathbf{x}) = \hat{\beta}_0 + \hat{\beta}_{11}g_{11}(\mathbf{x}) + \hat{\beta}_{12}g_{12}(\mathbf{x})$$

Other implementations begin with a best single hockey-stick function $g_1(\mathbf{x})$ in place of a pair, and $\hat{f}_1(\mathbf{x}) = \hat{\beta}_0 + \hat{\beta}_1g_1(\mathbf{x})$.

With a predictor $\hat{f}_l(\mathbf{x})$ in hand, a MARS implementation considers for addition to the set of basis functions being used, an additional reflected pair (or single hockey-stick function) or a pair (or single function) produced by multiplication of a reflected pair (or single function) by a basis function already being employed. This consideration will typically be subject to

- the constraint that no x_j appears in any candidate product more than some fixed number of times²
- an upper limit on the order of the products considered for inclusion

The best candidate pair (or single function) in terms of reducing SSE is added to the set of basis functions and a next predictor $\hat{f}_{l+1}(\mathbf{x})$ is fit by least squares.

²Allowing no x_j to appear more than once maintains the piece-wise linearity of sections/slices of the predictor.

According to some stopping criterion (typically phrased in terms of a minimum fractional reduction in $SSE = N \cdot \overline{err}$ or some inside-MARS "generalized cross-validation error" criterion) one ends forward selection with a suitable l no more than some user-specified maximum value. Some implementations then consider backward elimination of terms once forward selection is ceased.

Whatever "special sauce" is built into a particular MARS algorithm, user-supplied parameters ultimately determine the final predictor. Cross-validation must be applied *outside* the algorithm, optimizing a cross-validation error over choices of the parameters.

For example, if a particular implementation of MARS allows one to set **1)** an upper limit on the final number of terms in the model, **2)** an upper limit on the number of times that any x_j can appear in a product of hinge functions, and **3)** an upper limit on the order of any product of hockey stick functions, cross-validation and optimization across combinations of values of these variables (holding out one fold at a time and running the algorithm on the remainder).

The `train()` function in the `caret` package will cross-validate the MARS routine in the `earth` package across values of 1) and 3).

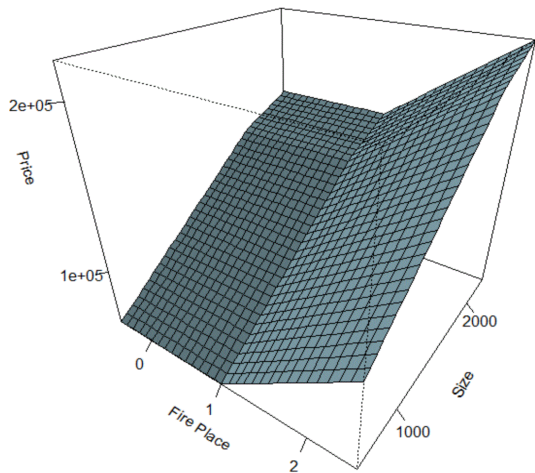
Running the `train` function in the `caret` package on the `earth` routine over all limits on numbers of terms in a final model 1 to 20, with maximum orders of model products 1 to 3 (using 8-fold cross-validation repeated 20 times) produced the choices of 4 terms in the final model with first order products only, and ultimately the predictor

$$\hat{f}(\mathbf{x}) = 146100 - 66.118 (1866 - \text{Size})_+ \\ + 37.124 (\text{Size} - 1866)_+ + 38204 (\text{Fireplace} - 1)_+$$

Since "Fireplace" (count) is discrete, picturing response as a function of continuous input variables is a bit artificial, but for purposes of illustrating the kinds of functions that are produced as MARS predictors, the plot on the next slide represents the function above.

MARS

MARS Predictor for Ames House Price Chosen by Cross-Validation in earth()



Another Direction

Additive Forms

The possibility of effective smoothing in 1 or 2 (or possibly 3) dimensions suggests the possibility of fitting predictors of the form

$$\hat{f}(\mathbf{x}) = \alpha + \sum_{j=1}^p g_j(x_j)$$

or even

$$\hat{f}(\mathbf{x}) = \alpha + \sum_{j=1}^p g_j(x_j) + \sum_{\substack{\text{some} \\ j, j'}} g_{jj'}(x_j, x_{j'})$$

to a training set with $\mathbf{x} \in \mathfrak{R}^p$ where the functions g_j and $g_{jj'}$ are arbitrary smooth functions of their univariate or bivariate arguments. (This can be done via algorithms that in sequence iteratively smooth residuals from a predictor including all but one term of such an additive form.)

Generalized Additive Models (GAMs)

More Sequential Model Building

The GAM forms on the previous slide are continuous-input versions of "main effects only" and "main effects plus some two-factor interactions" models of factorial analysis. Details of fitting and searching among such models for moderate-to-large p must become highly specialized and again amount to a kind of "special sauce" for any particular implementation of GAM fitting. But whatever the details of a particular algorithm, the tuning parameters of that algorithm can be chosen by cross-validation applied to the algorithm used on each remainder from a fold of the training set.

GAMs

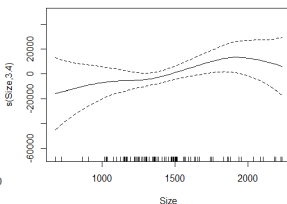
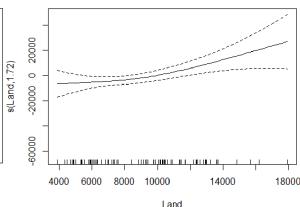
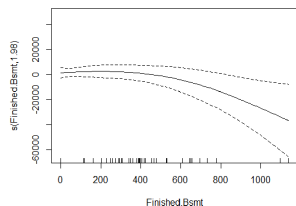
Ames House Price Example

The size of the Ames house price data set ($N = 88$) is small for fitting of even additive models in single input variables, let alone for trying to include smoothed functions of 2 or more variables in modeling. (GAM routines crash when trying to push the modeling beyond what is feasible for a given dataset size.) Applying the `train()` routine from the `caret` package with LOOCV to the `gam()` routine in the `gam` package, the variables "Finished Basement," "Basement Total," "Land" and "Size" are the only ones identified as candidates for smoothing (the other predictors enter a fitted GAM predictor linearly). When such a model is specified in `gam()` from the `mgcv` package, specifying spline smoothing returns non-linear components for 3 of the 4 variables. Finally, fitting a model linear in all predictors except "Finished Basement," "Land" and "Size" and allowing for arbitrary smooth versions of those variables produces what is represented on the next slides.

GAMs

Ames House Price Example Non-Linear Additive Predictor Components

The plots below provide non-linear smooth functions of the Finished Basement, Land, and Size variables that are more or less "automatically produced features"/transforms for the original variables.



Ultimately, the predictor fit by `gam()` from the `mgcv` package is

$$\begin{aligned}\hat{f}(\mathbf{x}) = & 20402 + 16730\text{Garage} + 1971\text{MutipleCar} + 1774\text{BedRooms} \\ & + 9033\text{CentralAir} + 12794\text{Fireplace} + 19140\text{FullBath} \\ & + 17125\text{HalfBath} + 44.85\text{BasementTotal} + 21074\text{BsmtBath} \\ & + 8058\text{Style2Story} - 1304\text{ZoneTownCenter} + g_1(\text{FinishedBsmt}) \\ & + g_2(\text{Land}) + g_3(\text{Size})\end{aligned}$$

The fitted smooth functions of Finished Basement, Land, and Size show the effects of changing one of those variables (with all others held fixed).

Statistical Machine Learning-6

Non-OLS Linear Predictors: Elastic Net, Ridge, and Lasso Regression

Stephen Vardeman

Analytics Iowa LLC

January 2018

Non-OLS Linear Predictors

Introduction

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

$$\hat{f}(\mathbf{x}) = \hat{\beta}_0 + \sum_{j=1}^p \hat{\beta}_j x_j$$

for an appropriate $\hat{\beta} \in \mathbb{R}^{p+1}$ than what is said in MLR (where ordinary least squares is used to fit the linear form to all p input variables or to some subset of M of them).

We next consider **non-OLS choices** of $\hat{\beta}$. *As in the spline smoothing material* for $p = 1$, we find that **penalization methods produce whole spectra of predictors of varying flexibilities, and cross-validation can be used to match predictor flexibility to training set information content.**

Non-OLS Linear Predictors

Framework

An alternative to seeking a suitable level of complexity in a linear prediction rule through subset selection and ordinary MLR is to employ a **shrinkage method** based on a penalized version of least squares to choose a vector $\hat{\beta}$. We consider a family of such methods, which has parameters that function as complexity measures and allow $\hat{\beta}$ to range between $\hat{\beta} = \mathbf{0}$ and $\hat{\beta} = \hat{\beta}^{\text{OLS}}$ depending upon complexity.

The outcomes of these methods are not equivariant to scaling used to express the input variables x_j . So we **assume that the output variable has been centered** (i.e. that $\sum y_i = 0$) **and that the columns of \mathbf{X} have been standardized** (and if originally \mathbf{X} had a constant column, it has been removed).

"Elastic Net" Regression

First Formulation (Unconstrained Optimization)

One type of penalized fitting of a linear predictor is so-called "**elastic net**" **regression**. For $\lambda_1 \geq 0$ and $\lambda_2 \geq 0$ the elastic net regression coefficient vector $\hat{\beta}_{\lambda_1, \lambda_2}^{\text{ENet}}$ minimizes over choice of $\beta \in \mathbb{R}^p$ the **penalized error sum of squares**

$$\sum_{i=1}^N \left(y_i - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda_1 \sum_{j=1}^p |\beta_j| + \lambda_2 \sum_{j=1}^p \beta_j^2 \quad (1)$$

The $\lambda_1 = 0$ version of this is called "**ridge**" **regression**, the $\lambda_2 = 0$ version is "**lasso**" (least absolute selection and shrinkage operator) **regression**, and when both $\lambda_1 = 0$ and $\lambda_2 = 0$ this is ordinary least squares MLR.

Elastic Net Regression

Equivalent Formulation (Constrained Optimization)

The elastic net unconstrained minimization problem has an equivalent *constrained minimization* description. That is, for $\gamma \in [0, 1]$ and $t > 0$, the coefficient vector $\hat{\beta}_{\gamma,t}^{\text{ENet}}$ minimizes $SSE(\beta) = \sum_{i=1}^N (y_i - \sum_{j=1}^p \beta_j x_{ij})^2$ over choice of $\beta \in \mathbb{R}^p$ with

$$\sum_{j=1}^p (\gamma |\beta_j| + (1 - \gamma) \beta_j^2) \leq t$$

For every (λ_1, λ_2) pair producing coefficient vector $\hat{\beta}_{\lambda_1, \lambda_2}^{\text{ENet}}$ there is a pair (γ, t) producing $\hat{\beta}_{\gamma,t}^{\text{ENet}} = \hat{\beta}_{\lambda_1, \lambda_2}^{\text{ENet}}$ (and the same value of SSE) and vice versa. The choice $\gamma = 1$ produces the **lasso constraint** $\sum_{j=1}^p |\beta_j| \leq t$ and the choice $\gamma = 0$ produces the **ridge constraint** $\sum_{j=1}^p \beta_j^2 \leq t$.

Elastic Net Regression

Penalization and "Shrinkage"

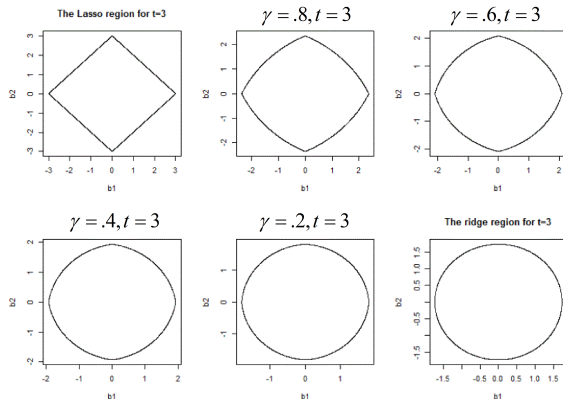
Large t (small λ_1 and λ_2) is little penalization. Small t (large λ_1 or λ_2) is large penalization. **The overall effect of penalization is to "shrink" the ordinary least squares coefficient vector towards the origin in \mathbb{R}^p .**¹ In turn, this implies that centered predictions $\hat{f}(\mathbf{x}_i)$ are shrunken towards 0 and **uncentered predictions are shrunken towards \bar{y} .**

More qualitative insight concerning the nature of elastic net shrinkage can be had from consideration of the geometry of constraint regions. Some representations of $p = 2$ constraint regions for $t = 3$ (made using some code of Prof. Huaqing Wu) are on the next slide. As γ goes from 1 to 0 the elastic net constraint region goes from (rotated) square to circular.

¹This is completely obvious in the constrained formulation, in that small t means that $\hat{\beta}_{\gamma,t}^{\text{ENet}}$ (which is inside the constraint region) must be small.

Elastic Net Regression

Geometry of Constraint Regions



The obvious "corners" on elastic net constraint regions for γ near 1 have an interesting and attractive consequence.

The Lasso

Geometry of Lasso Optimization

Below is a representation of the $p = 2$ constrained optimization problem solved by the **lasso** coefficient vector, $\hat{\beta}_{0,t}^{\text{lasso}}$. **Corners on the constraint region make it typical for some coordinates of the lasso coefficient vector to be exactly 0.**

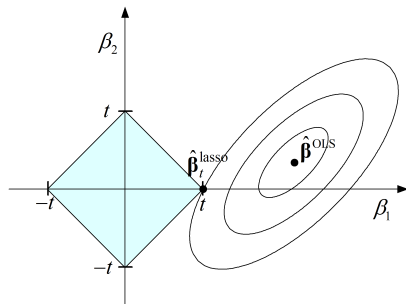


Figure: Contours of $SSE(\beta)$, the constraint region, and the OLS and lasso fitted vectors.

Ridge Regression

Geometry of Ridge Optimization

In contrast is the representation below of the problem solved by the **ridge** coefficient $\hat{\beta}_{0,t}^{\text{ridge}}$ for $p = 2$. **The lack of corners on the constraint region makes exactly 0 fitted coefficients rare.**

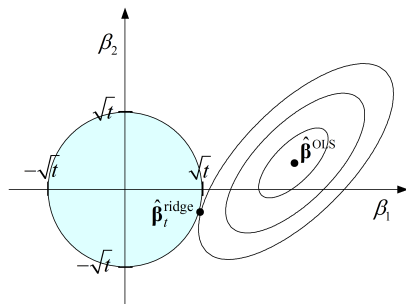


Figure: Contours of $SSE(\beta)$, the constraint region, and the OLS and ridge fitted vectors.

Elastic Net Regression

Computation, Etc.

Only for the ridge case are there explicit formulas for elastic net regression coefficients² and in general, numerical optimization is required. The cartoons on the previous two slides make it obvious that entries of an elastic net coefficient vector do not necessarily have the same sign as the corresponding entry of the OLS coefficient vector (and the sign can change with values of the penalty parameters). The elastic net coefficient vector changes continuously in the penalty parameters, and it is only in an overall sense that $\hat{\beta}_{\gamma,t}^{\text{ENet}}$ converges monotonically to $\mathbf{0}$ as $t \rightarrow 0$. Individual entries of the coefficient vector are not necessarily monotone in t . And in cases where "corners" produce exactly 0 fitted coefficients, because an entry is 0 for a particular t does not necessarily imply that it is 0 for smaller t .

²An explicit matrix form for the ridge regression coefficient vector is $\hat{\beta}_{\lambda}^{\text{ridge}} = (\mathbf{X}'\mathbf{X} + \lambda\mathbf{I})^{-1} \mathbf{X}'\mathbf{Y}$. This is highly reminiscent of the coefficient vector applied to \mathbf{H} to produce smoothing spline predictions.

The Elastic Net

glmnet (Unconstrained Optimization) Parameterization

A different parameterization of the unconstrained elastic net optimization criterion (1) used in the `glmnet` package is (for $\lambda \geq 0$ and $0 \leq \alpha \leq 1$) that $\hat{\beta}_{\lambda, \alpha}^{\text{ENet}}$ be a vector $\beta \in \mathbb{R}^p$ minimizing

$$\frac{1}{2} \cdot \frac{1}{N} \text{SSE}(\beta) + \lambda \left(\alpha \sum_{j=1}^p |\beta_j| + \frac{(1-\alpha)}{2} \sum_{j=1}^p \beta_j^2 \right) \quad (2)$$

It is easy enough to work out the relationships between parameter vectors (λ_1, λ_2) and (λ, α) above. The pair (λ, α) clearly corresponds to

$$\lambda_1 = 2N\lambda\alpha \quad \text{and} \quad \lambda_2 = N\lambda(1-\alpha)$$

in formulation (1). On the other hand, a bit of algebra shows that the pair (λ_1, λ_2) there corresponds to

$$\lambda = \frac{\lambda_1 + 2\lambda_2}{2N} \quad \text{and} \quad \alpha = \frac{\lambda_1}{\lambda_1 + 2\lambda_2}$$

here.

The Elastic Net

Choice of Parameters

Fixing attention (wolog) on the specification of an elastic net predictor corresponding to form (2), **the ridge class of predictors is the $\alpha = 0$ version of the elastic net and the lasso class is the $\alpha = 1$ sub-class.** So choosing a best elastic net predictor by cross-validation over values of both α (that controls how the penalty is apportioned between lasso and ridge parts) and λ (that in governs the overall strength of the penalization) will do at least as well as is possible considering only ridge or lasso predictors.

The `train()` routine in the `caret` package will optimize cross-validation errors across both α and λ , calling the `glmnet` routine (searching over a user-specified grid of (α, λ) pairs).

The Elastic Net

Cross-Validation on the Ames House Price Data

Some fairly extensive searching across (α, λ) pairs (using averages of 100 repeats of 8-fold cross-validations) with the caret package `train()` routine has left your instructor convinced that the Ames House Price example is one in which *nothing is to be gained from using a lasso penalty component in the elastic net*. The best combination he was able to locate had $\alpha = 0$ and $\lambda = 14000$. This set of parameters can then be used in `glmnet` to produce a ridge regression predictor of y .

The Elastic Net

Perspective

The elastic net accomplishes in continuous fashion what is attempted in a more ad hoc discrete way when one looks for good sub-models of a large p MLR model. Unless one forces use of its lasso specialization, cross-validation with it does not often completely eliminate many (if any) variables x_j from consideration. In that sense its usefulness as a "subset selection" tool is perhaps oversold. But what it can do well is shrink predictions based on a large number of features in a way that prevents over-fit. One is then left with a good predictor that is nevertheless based on a large p . This is potentially especially helpful in contexts where a large number of basis functions are employed to flexibly model a relationship between \mathbf{x} and y .

Statistical Machine Learning-7

Regression Trees: Part 1 of CART (Classification and Regression Trees)

Stephen Vardeman

Analytics Iowa LLC

January 2018

Tree Predictors

Binary Splitting

Tree predictors aim to split an input space \mathcal{R}^p into **simple p -dimensional "rectangular" regions within which output values for training cases (y_i s) are relatively homogeneous.**

Sequential binary splitting of existing rectangles (splitting a single existing rectangle in a tree on a single coordinate of \mathbf{x} at each step of a search) is employed. This is not because it is in any sense optimal but because it is feasible to do, can be easily interpreted, and often gives excellent results. This strategy is a **forward-selection/"greedy" method** for developing a predictor constant on p -dimensional rectangles, since each split is made without "looking ahead" at what could be achieved in a future step if a particular not-presently-advantageous split were made immediately.

Regression Trees

Binary Splitting ($p=2$ Case for Concreteness)

For example, with $p = 2$, one begins with a 2-D rectangle $[a, b] \times [c, d]$ defined by

$$a = \min_{i=1,2,\dots,N} x_{i1}, b = \max_{i=1,2,\dots,N} x_{i1}, c = \min_{i=1,2,\dots,N} x_{i2} \text{ and } d = \max_{i=1,2,\dots,N} x_{i2}$$

and looks for a way to split it at $x_1 = s_1$ or $x_2 = s_1$ so that the resulting two rectangles minimize

$$SSE = \sum_{\text{rectangles}} \sum_{\substack{i \text{ with } \mathbf{x}_i \text{ in} \\ \text{the rectangle}}} (y_i - \bar{y}_{\text{rectangle}})^2$$

One then splits (optimally) one of the (now) two rectangles at $x_1 = s_2$ or $x_2 = s_2$, etc.

Regression Trees

Predictor and Training Error

Where l rectangles in \mathbb{R}^p have been created (through $l - 1$ splits), and

$R(\mathbf{x})$ = the rectangle to which \mathbf{x} belongs

the tree predictor is

$$\hat{f}_l(\mathbf{x}) = \frac{1}{\# \text{ training input vectors } \mathbf{x}_i \text{ in } R(\mathbf{x})} \sum_{i \text{ with } \mathbf{x}_i \in R(\mathbf{x})} y_i$$

and the training error is $1/N$ times

$$SSE = \sum_{i=1}^N (y_i - \hat{f}_l(\mathbf{x}_i))^2$$

To continue splitting beyond l rectangles, one of the existing rectangles is split at a value s_j on some coordinate x_j to produce the greatest possible reduction in SSE . While this is sensible, there is no guarantee that after l splits the best (in terms of SSE) possible set of $l + 1$ rectangles in \mathbb{R}^p has been produced.

Regression Trees

A Small $p=2$ Hypothetical Case

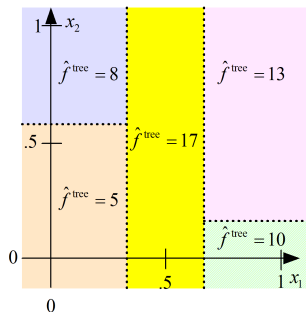
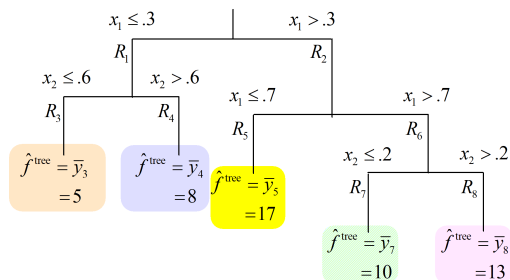
For sake of illustration, suppose that in a $p = 2$ problem $0 < x_{j1} < 1$ and $0 < x_{j2} < 1$ for all N training cases and:

1. a first split is made on x_1 at $.3$, producing rectangles R_1 where $x_{j1} \leq .3$ and mean output is $\bar{y}_1 = 7$ and R_2 where $x_{j1} > .3$ and mean output is $\bar{y}_2 = 15$,
2. a second split is made by splitting R_1 on x_2 at $.6$, creating (sub-)rectangles R_3 where $x_{j2} \leq .6$ and mean output is $\bar{y}_3 = 5$ and R_4 where $x_{j2} > .6$ and mean output is $\bar{y}_4 = 8$,
3. a third split is made by splitting R_2 on x_1 at $.7$, creating (sub-)rectangles R_5 where $x_{j1} \leq .7$ and mean output is $\bar{y}_5 = 17$ and R_6 where $x_{j1} > .6$ and mean output is $\bar{y}_6 = 12$,
4. a final split is made by splitting R_6 on x_2 at $.2$, creating (sub-)rectangles R_7 where $x_{j2} \leq .2$ and mean output is $\bar{y}_7 = 10$ and R_8 where $x_{j2} > .2$ and mean output is $\bar{y}_8 = 13$.

Regression Trees

Representations of the Small $p=2$ Hypothetical Case

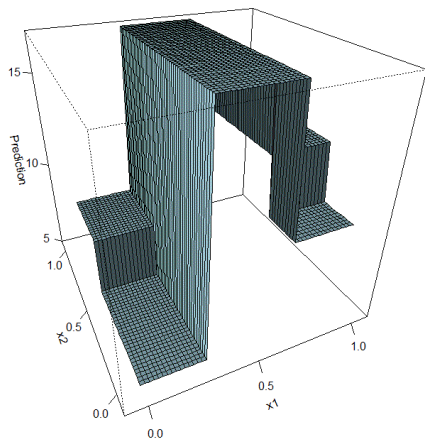
Below are two representations of the hypothetical $p = 2$ tree predictor.



Regression Trees

Perspective Plot for the Small $p=2$ Hypothetical Case

Below is a perspective plot companion to the representations on the previous slide.



Regression Trees

Choosing Tree Complexity and Cross-Validation

So how does one find a tree of appropriate complexity? One direct possibility is to consider only trees met in forward-selection-tree-making, and treat the number of splits/final nodes in the tree as a complexity parameter. Cross-validation can be used to compare numbers of splits and thus choose one for use with the whole training set. This can be done using the `train()` function in `caret`. But something better is possible.

One can build a large tree in forward-selection fashion and then efficiently find a **nested sequence of "pruned" optimal sub-trees of the large tree**

1. parameterized by another **complexity parameter**, and
2. potentially containing sub-trees *better* than ones of the same number of nodes met in the forward-selection process.

"Grow and prune to a given complexity" can then be subjected to **cross-validation**.

Regression Trees

Penalized Error Sum of Squares and Pruning

For T a sub-tree of some fixed large tree T_0 (e.g. grown until splitting any rectangle optimally would put less and 5 training cases \mathbf{x}_i in one resulting part) and $\lambda > 0$ define the penalized error sum of squares

$$C_\lambda(T) = SSE(T) + \lambda |T|$$

(for $|T|$ the number of final nodes in T and $SSE(T)$ the error sum of squares for the predictor). Write $T(\lambda)$ for the sub-tree of T minimizing $C_\lambda(T)$ and let \hat{f}_λ be the corresponding predictor.

How to find a sub-tree $T(\lambda)$ optimizing $C_\lambda(T)$ (without making an exhaustive search over sub-trees for every different value of λ) has a workable answer. There is a relatively small number of nested candidate sub-trees that are the only ones that are possible minimizers of $C_\lambda(T)$, and as λ increases, $T(\lambda)$ moves through that sequence of sub-trees from the largest/original tree to the smallest.

Regression Trees

Optimal Sub-trees

While we won't give details of exactly how the nested sequence of sub-trees is produced, they are not particularly hard. The function of λ ,

$$C_\lambda(T_\lambda) = \min_T C_\lambda(T)$$

and the optimizing nested sequence of sub-trees can be computed very efficiently. λ is a **complexity parameter** with $|T(\lambda)|$ decreasing in λ .¹ It parameterizes good elements of a much larger set of trees than are reached in forward selection tree-building *and is similar in spirit to the complexity parameters associated with spline smoothing and penalization methods like elastic net prediction*. It is an excellent guide through the very large class of binary tree predictors.

¹As the continuous variable λ goes from 0 to ∞ there will typically be integer values "skipped" by $|T(\lambda)|$. Not every possible number of final nodes has a corresponding sub-tree optimizing the penalized training error C_λ .

Regression Trees

Cross-Validation and Choice of a Tree

Cross-validation based on cost-complexity can proceed as follows. For each of K remainders ($\mathbf{T} - \mathbf{T}_k$ in the notation of the original description of cross-validation)

1. grow a tree on $\mathbf{T} - \mathbf{T}_k$ as far as possible subject to the final node with the fewest training \mathbf{x}_i containing at least 5 or less such points, then
2. "prune" the tree in 1. back by for each $\lambda > 0$ minimizing over choices of sub-trees, the quantity

$$C_{\lambda}^k(T) = SSE_k(T) + \lambda |T|$$

(for T a candidate tree and $SSE_k(T)$ the $\mathbf{T} - \mathbf{T}_k$ error sum of squares for the corresponding predictor). Writing $T_k(\lambda)$ for the sub-tree minimizing $C_{\lambda}^k(T)$ let \hat{f}_{λ}^k be the corresponding predictor.

Regression Trees

Cross-Validation Choice of Tree

3. Then (as in the original description of cross-validation), letting $k(i)$ be the index of fold \mathbf{T}_k containing training case i , and compute the cross-validation error

$$CV(\lambda) = \frac{1}{N} \sum_{i=1}^N \left(\hat{f}_{\lambda}^{k(i)}(\mathbf{x}_i) - y_i \right)^2$$

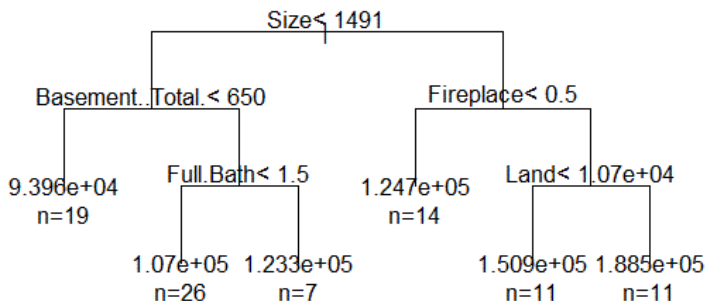
For $\hat{\lambda}$ a minimizer of $CV(\lambda)$, one then operates on the entire training set, growing a tree as far as possible, subject to the final node with the fewest training \mathbf{x}_i containing at least 5 training cases, then finding the sub-tree, say $T(\hat{\lambda})$, optimizing $C_{\hat{\lambda}}(T) = SSE(T) + \hat{\lambda}|T|$, and using the corresponding predictor $\hat{f}_{\hat{\lambda}}$.

The `train()` function in `caret` will do this cross-validation and selection of $\hat{\lambda}$ based on the `rpart` tree-building package.

Regression Trees

Ames Housing Example

The `train()` function in `caret` was used to cross-validate and optimize tree predictors of Ames House Price, using 100 repeats of 8-fold cross-validation. The complexity value of $\lambda \approx .007$ is indicated. Using the whole training set with this complexity parameter, a tree with 6 nodes is produced. This is illustrated below.



Regression Trees

Comments on Transformations and on Ordinal Inputs

The rectangle-building process used in making a regression tree is such that **making monotone transformations of any or all of the coordinates of an input x have no effect on the ultimate predictor.** (That is, all inputs would have the same predictions with or without transformation before tree-building.) So in some sense **the regression tree idea simplifies predictor-building by eliminating the question of whether some other scale should be used to express a given input x_j .**

This fact also means that trees handle **ordinal input variables** in a way that preserves and employs orderings of their values without making unjustifiable assignments of scale to those values (that potentially prevent effective use of the information such inputs carry).

Regression Trees

More Perspective

- CART technology (at least where a reasonably small tree is ultimately produced) has the very strong point of being highly interpretable by even very non-quantitative consumers of data analytics
- The tree-growing methods here employ "one-step-at-a-time"/ "greedy" (unable to defer immediate reward for the possibility of later success) methods. They are not guaranteed to follow paths through the set of trees that ever get to "best" ones since they are "**myopic**"/"**greedy**" never considering what "might be" later in a search if a current step were taken that provides little immediate payoff
- Conventional wisdom is that the tree splitting algorithm "favors" splitting on continuous input variables over splitting on values of discrete ones

Statistical Machine Learning-8

Predictors Built on Bootstrap Samples: Bagging and Random Forests

Stephen Vardeman

Analytics Iowa LLC

January 2018

Bootstrap Samples and "Bagging"

"Resampling" the Training Set Repeatedly, Fitting, and Averaging

A way to try to prevent a prediction methodology from producing \hat{f} "too sensitive" to exact characteristics of a training sample is to employ "**bootstrapping**." This involves some large number, B , of "bootstrap" samples of size N from the training set \mathbf{T} . Each of these, $\mathbf{T}_1^*, \mathbf{T}_2^*, \dots, \mathbf{T}_B^*$, is a random sample *with replacement* of size N from \mathbf{T} . Applying a fixed method of prediction B times produces for each $b = 1, \dots, B$

predictor \hat{f}^{*b} based on \mathbf{T}_b^*

"Bootstrap aggregation" or "**Bagging**" for SEL is then the use of

$$\hat{f}_{\text{bag}}^B(\mathbf{x}) \equiv \frac{1}{B} \sum_{b=1}^B \hat{f}^{*b}(\mathbf{x})$$

The hope is to average (not-perfectly-correlated as they are built on not-completely-overlapping bootstrap samples) low-bias/high-variance predictors to reduce variance (while maintaining low bias).

(Large B) Convergence of a Bagged Predictor

The Limiting Bagging Predictor

Even for fixed training set \mathbf{T} and input \mathbf{x} a bagging predictor $\hat{f}_{\text{bag}}^B(\mathbf{x})$ is *random* (varying with the selection of the bootstrap samples). Let E^* denote averaging over the creation of a single bootstrap sample and \hat{f}^* be the predictor derived from such a bootstrap sample. Then

$$E^* \hat{f}^*(\mathbf{x}) = \hat{f}_{\text{bag}}(\mathbf{x})$$

is the "true"/large- B bagging predictor¹ with simulation-based approximation $\hat{f}_{\text{bag}}^B(\mathbf{x})$.

($\hat{f}_{\text{bag}}^B(\mathbf{x}) \rightarrow E^* \hat{f}^*(\mathbf{x}) = \hat{f}_{\text{bag}}(\mathbf{x})$ as $B \rightarrow \infty$ by the law of large numbers.)

¹Unless the operations applied to a training set to produce \hat{f} are linear, $E^* \hat{f}^*(\mathbf{x})$ will differ from the predictor computed from the full training data, $\hat{f}(\mathbf{x})$.

Out of Bag Error

Cases Missed in a Bootstrap Sample from the Training Set

Any particular bootstrap sample \mathbf{T}_b^* fails to contain (on average) about 37% of training cases, that might be called an **out-of-bag (OOB) sample**. OOB samples can serve as test sets for bagged predictors. More precisely, it is common to make (and plot versus B) a **running-cross-validation estimate of error** (Err) based on out-of-bag (OOB) samples.

That is, suppose that for each b the set of indices $I(b) \subset \{1, 2, \dots, N\}$ for which the corresponding training cases are OOB (not included in the bootstrap training set \mathbf{T}_b^*). Let

$$\hat{y}_{iB}^* = \frac{1}{\# \text{ of indices } b \leq B \text{ such that } i \in I(b)} \sum_{b \leq B \text{ such that } i \in I(b)} \hat{f}^{*b}(\mathbf{x}_i)$$

Then an estimate of Err for the bagged predictor \hat{f}_{bag} is

$$\text{OOB}(B) = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_{iB}^*)^2$$

Plotting the OOB Error

Convergence of the Running Estimate of Err

As B increases, one can expect $\hat{f}_{\text{bag}}^B(\mathbf{x})$ to better approximate its limit $\hat{f}_{\text{bag}}(\mathbf{x})$ and $\text{OOB}(B)$ to better approximate Err for $\hat{f}_{\text{bag}}(\mathbf{x})$. So plotting $\text{OOB}(B)$ versus B and determining when B is large enough that $\text{OOB}(B)$ seems to have leveled off at some limiting value is a common way of determining when both 1) the extra/non-intrinsic noise introduced into the creation of a predictor by the bootstrap sampling has been averaged away and 2) a reliable measure of efficacy for the bagged predictor has been arrived at.

Bagging Trees

Trees and Trees With Random Restrictions on Splits

Bagging can in principle be applied to any form of SEL predictor. But in practice, by far the most common application is to the case where the base predictors that are bagged are trees. And by far the most common version of bagged tree predictor is the so-called "**random forest**," that involves an unexpected variant of the kind of tree-building discussed thus far. Namely, for every contemplated splitting of a rectangle, only a (newly) randomly chosen subset of the p inputs are considered for splitting. **A random forest is then a bagged version of a special (randomized) tree.**

Random Forest Specifics

Single Special Trees and Their Bagged Version

With bootstrap samples $\mathbf{T}_1^*, \mathbf{T}_2^*, \dots, \mathbf{T}_B^*$, for each \mathbf{T}_b^* a corresponding regression tree is made by

1. at each node, randomly selecting m of the p input variables and finding a 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, n_{\min} , of training cases.

Let $\hat{f}^{*b}(\mathbf{x})$ be the corresponding tree-based predictor.

A random forest predictor is then the bagged (specialized) tree predictor

$$\hat{f}_{\text{RF}}^{*B}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^B \hat{f}^{*b}(\mathbf{x})$$

Random Forests

Tuning/Complexity Parameters

The basic tuning parameters in the development of $\hat{f}_{\text{RF}}^{*B}(\mathbf{x})$ are m , and n_{\min} , and (if used) a maximum tree depth. $m = p$ is the case of bagging ordinary trees (with no restriction on which variables are candidates for splitting). Standard default values of parameters are $m = \lfloor p/3 \rfloor$ and $n_{\min} = 5$, but they can be chosen to minimize the (large B) OOB error.

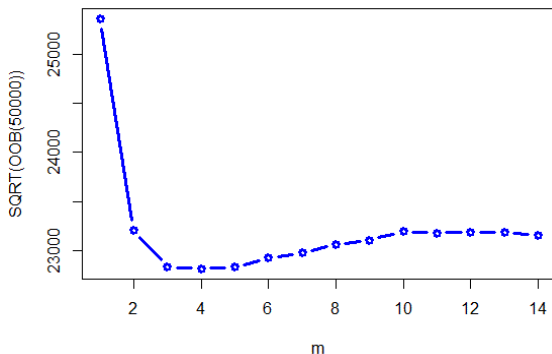
Bagging provides its own "internal" version of cross-validation and there is no need to wrap another cross-validation around a random forest in order to approximate Err.

In spite of the fact that for small B the (random) predictor \hat{f}_{RF}^{*B} is built on a small number of trees and is fairly simple, B is *not* really a complexity parameter, but is rather a *convergence* parameter that governs convergence of \hat{f}_{RF}^{*B} to some (law of large numbers) limit \hat{f}_{RF} and $\text{OOB}(B)$ to Err.

Random Forests

Ames House Price Example: Choice of m

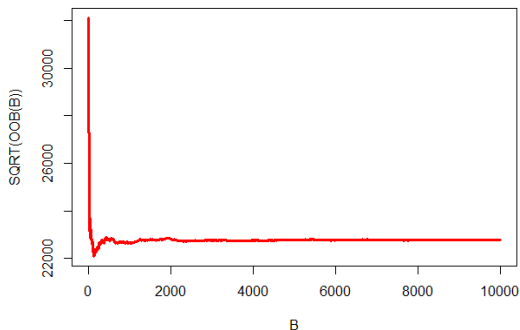
The `randomForest` package was used to fit random forests to the Ames House Price data for $m = 1, 2, \dots, 14$ (with all other parameters at their default values). A plot of the square root of the OOB error based on $B = 50000$ trees is below. The best value of m is 4 with $\sqrt{\text{OOB}(50000)} \approx 22813$.



Random Forests

Ames House Price Example: Plot of $\text{OOB}(B)$ for $m=4$

Below is a plot of $\sqrt{\text{OOB}(B)}$ versus B for $m = 4$ in the Ames House Price example computations presented on the previous slide. (Only values for $B = 5$ to $B = 10000$ are plotted.) It shows the convergence of $\text{OOB}(B)$ far before the value of $B = 50000$ used to make the choice of m .



Random Forests

Ames House Price Example Additional Details

Varying both m and n_{\min} in `randomForest` applied to the prediction of Price produces the values below for $\sqrt{\text{OOB}(50000)}$ (rows correspond to values of m and columns to values of n_{\min}). The performance of a random forest in the problem is reasonably insensitive to the choice of $m \geq 2$. The default values of $m = \lfloor 14/3 \rfloor = 4$ and $n_{\min} = 5$ can be improved upon only slightly, the completely optimal choice being $m = 3$ and $n_{\min} = 1$.

	OOB1	OOB2	OOB3	OOB4	OOB5	OOB6	OOB7	OOB8
1	25105	25163	25162	25277	25362	25473	25588	25759
2	22937	22922	23030	23142	23208	23348	23458	23559
3	22670	22681	22729	22763	22829	22918	23033	23086
4	22678	22692	22743	22716	22813	22870	22939	22998
5	22777	22780	22782	22819	22829	22876	22955	23033
6	22859	22880	22882	22904	22924	22964	22987	23044
7	22970	22994	22957	22991	22975	23036	23031	23069
8	23055	23061	23080	23031	23063	23047	23069	23141
9	23148	23170	23112	23131	23102	23110	23121	23173
10	23165	23183	23172	23188	23192	23148	23163	23162
11	23207	23248	23230	23196	23177	23173	23165	23190
12	23222	23226	23215	23214	23188	23174	23174	23205
13	23195	23218	23229	23166	23181	23168	23168	23163
14	23190	23224	23203	23177	23150	23194	23157	23121

Random Forests

??Over-Fitting??

There is a fair amount of confusing discussion in the literature about the impossibility of a random forest "over-fitting" with increasing B . This seems to be related to test error *not* initially-decreasing-but-then-increasing-in- B (which is perhaps loosely related to $\text{OOB}(B)$ converging to a positive value associated with the limiting predictor \hat{f}_{RF} , and not to 0). But as HTF point out on their page 596, it is an entirely different question as to whether \hat{f}_{RF} itself is "too complex" to be adequately supported by the training data, \mathbf{T} . And the whole discussion seems very odd in light of the fact that B is a convergence parameter, NOT a complexity parameter. Complexity must properly refer to the limiting \hat{f}_{RF} not to the approximation \hat{f}_{RF}^{*B} .

It is certainly possible that a random forest is more complex than a training set can support. The result in such cases is Err and the limiting $\text{OOB}(B)$ being unnecessarily large.

Random Forests

???Correlation Between Trees???

There is also a fair amount of confusing discussion in the literature about the role of the *random* selection of the m predictors to use at each node-splitting (and the choice of m) in reducing "correlation between trees in the forest." The Breiman/Cutler web site http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm says that the "forest error rate" (presumably Err for \hat{f}_{RF}) depends upon "the correlation between any two trees in the forest" and the "strength of each tree in the forest."

Any precise/technical meaning of these is not clear. In rough/vague terms, increasing m increases both similarity between and flexibility of the individual trees, the first degrading Err and the second improving it.

Statistical Machine Learning-9

SEL Boosting and Stacking

Stephen Vardeman

Analytics Iowa LLC

January 2018

Combinations of "Ensembles" of Predictors

Boosting and Stacking

Bagging averages an "ensemble" of predictors (consisting of versions of a single predictor computed from different bootstrap samples) to make a predictor \hat{f}_{bag} .

Here we consider other important versions of the basic idea that multiple predictors might be in some way combined to make a single effective SEL predictor. The primary motivation of the first, so-called "**boosting**," is the reduction of model bias through a kind of successive approximation to a conditional mean $E[y|\mathbf{x}]$ function. The second, so-called "**stacking**," seeks a good function of several predictors (a "meta-predictor" or "super-learner") that might be better than any single one in the ensemble.

SEL Boosting

Motivation

There is a general "gradient boosting" method due to Friedman. Here we consider the SEL special case of Friedman's gradient boosting. The idea is to sequentially build a good approximator for $E[y|\mathbf{x}]$ by successively adding small corrections (based on modeling residuals) to current approximators. This is parallel to many "**successive approximation**" methods of numerical analysis for root finding and optimization that repeatedly correct a current approximate root or optimizer.

SEL Boosting

An Algorithm

A SEL boosting algorithm is:

1. Start with $\hat{f}_0(\mathbf{x}) = \bar{y}$.
2. With $\hat{f}_{m-1}(\mathbf{x})$ in hand and possible model for $y - \hat{f}_{m-1}(\mathbf{x})$ available, say $\beta_m h_m(\mathbf{x}, \gamma)$ for some given form $h_m(\cdot, \cdot)$ and unknown parameters β_m and γ_m , fit $\hat{\beta}_m$ and $\hat{\gamma}_m$ by least squares. (This is approximation of the gradient of $SSE (N \cdot \overline{\text{err}} \text{ for SEL})$ wrt $\hat{\mathbf{Y}}$.)
3. For some "learning rate" $\nu \in (0, 1)$ set

$$\hat{f}_m(\mathbf{x}) = \hat{f}_{m-1}(\mathbf{x}) + \nu \hat{\beta}_m h_m(\mathbf{x}, \hat{\gamma}_m)$$

(this is an approximate gradient descent step for SSE) and return to step 2.

One iterates on m from 1 through some number of iterations, M . The smaller is ν , the larger must be M for effective boosting.

SEL Boosting

Comments

SEL boosting successively corrects a current predictor by adding to it some fraction of a predictor for its residuals. The value ν functions as a **complexity** or regularizing **parameter**, as does M . (Small ν together with large M correspond to large complexity.) Boosting ends with a linear combination of fitted forms as a final predictor/approximator for $E[y|\mathbf{x}]$.

Sequential modification of a predictor is not discussed in ordinary regression/linear models contexts because if a base predictor is an OLS predictor for a fixed linear model, corrections to an initial fit based on this same model fit to residuals will predict that all residuals are 0. In this circumstance boosting does nothing to change or improve an initial OLS fit.

SEL Boosting

Ames House Price Example

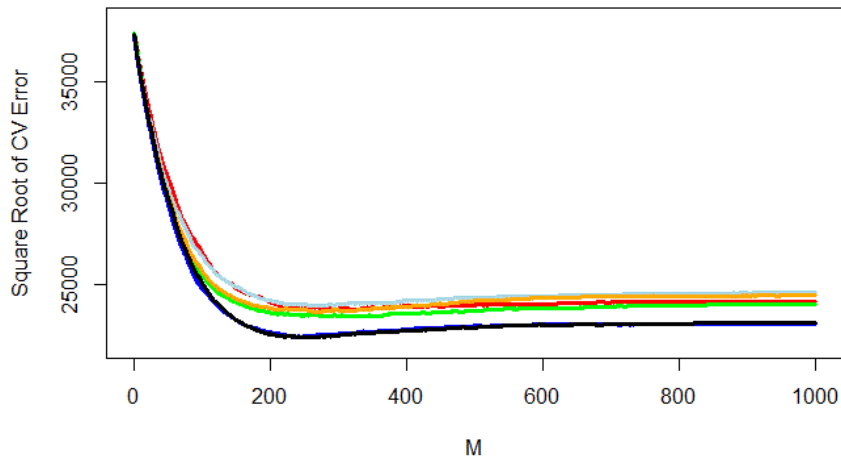
The R package `gbm` does SEL boosting with regression trees. It can be used in the `caret` function `train()` to produce a cross-validated choice of sum of regression trees. Some work with these routines on the Ames house price data produced a CV RMSE of 22236 for a set of parameters $M = 125$ and $\nu = .04$ for boosting with trees with $n_{\min} = 6$ and a maximum tree depth of 2. These parameters produce an 8-fold cross-validation error comparable to the best random forest OOB error (22670) previously identified for this problem.

Allowing M to be too large for a given ν will result in overfitting. This can be avoided by taking account of cross-validation errors. The plot on the next slide is based on the `gbm` package's option for producing cross-validation errors. It shows several plots of the square root of 8-fold cross-validation errors for regression trees made using $\nu = .01$, $n_{\min} = 2$ and a maximum tree depth of 20 and makes clear the negative effect of using M that is too large.

SEL Boosting

Ames House Prices and the Effect of M on CV Error in Boosting

CV Error for Several 8-Fold Splits



Boosting is a big deal. It is increasingly recognized as **the single most widely effective method of predictor development.** That should be unsurprising. To the extent that it is recognized as successive approximation from numerical analysis (that is widely effective more or less regardless of the form of a function being optimized) some version of it *should be* effective in essentially any prediction problem. The "XGBoost" implementation of gradient boosting has become wildly popular and is widely seen as "the silver bullet" in public predictive analytics contests.

Blanket enthusiasm for boosting down-plays difficulties in its implementation. Its most effective versions have many tuning parameters, and without careful cross-validation employed to guide their choice, overfitting is almost guaranteed. But the burden already implicit in cross-validation with a fixed set of parameters combined with a huge grid of multiple values of multiple parameters to be compared can produce enormous computational loads.

(Ordinary) Stacking

Linear Combinations of M Predictors

Suppose that M predictors (all based on the same training data) are available, $\hat{f}_1, \hat{f}_2, \dots, \hat{f}_M$. Under squared error loss, one might seek a coefficient/weight vector \mathbf{w} for which the predictor

$$\hat{f}(\mathbf{x}) = w_0 + \sum_{m=1}^M w_m \hat{f}_m(\mathbf{x}) \quad (1)$$

is effective. Why this can improve on any single one of the \hat{f}_m s is "obvious," since the set of possible \mathbf{w} includes vectors with one entry $w_m = 1$ and all others 0. As linear form (1) is inherently more flexible than any one of its constituent predictor forms, it potentially provides important reduction of model bias and improved overall prediction.

(Generalized)

Stacking/"Meta-Predictors"/"Super-Learners"

Other Functions of M Predictors

One important way to view the stacked predictor (1) is as a linear predictor based on M new "features" that are the values of the ensemble. That suggests applying some standard predictor methodology to a "training set" consisting of M vectors of predictions ... *with or without some or all of the original input variables also reused as inputs*. In a relatively simple case where no original inputs are "reused" the generalization of (1) is

$$\tilde{f}(\mathbf{x}) = \hat{f}(\hat{f}_1(\mathbf{x}), \hat{f}_2(\mathbf{x}), \dots, \hat{f}_M(\mathbf{x})) \quad (2)$$

for some appropriate prediction algorithm \hat{f} . As this is more general than "ordinary" stacking, it has the potential to be even more effective than a linear combination of the M predictors could be.

Stacking is a big deal. From the earliest of the public predictive analytics contests (the Netflix Prize contest run 2006-2009) it has been common for winning predictions to be made by "end-of-game" merging of effort by two or more separate teams that in some way average their separate predictions. More and more references are made on contest forums to various strategies for combining basic predictors. Multiple-level versions of the stacking structure are even discussed (though in truth, they are but structured versions of the general form (2)).

While the success of some (?luckiest among a number of?) ad hoc choices of stacking forms in particular situations is undeniable, principled choices of forms and parameters for f in (2) involve both logical subtleties and huge computational demands.

Stacking/Super-Learners

Perspective

A standard recommendation for principled choice of coefficient vector \mathbf{w} for predictor (1) is to find the vector $\mathbf{w}^{\text{stack}}$ minimizing over \mathbf{w} the sum

$$\sum_{i=1}^N \left(y_i - w_0 - \sum_{m=1}^M w_m \hat{f}_m^i(\mathbf{x}_i) \right)^2$$

for \hat{f}_m^i "the m th predictor fit to $\mathbf{T} - \{(\mathbf{x}_i, y_i)\}$, the training set with the i th case removed." ($\mathbf{w}^{\text{stack}}$ optimizes a kind of leave-one-out cross-validation error for a linear combination of \hat{f}_m 's.)

A potential logical and computational concern here is that in this description \hat{f}_m^i cannot depend upon (\mathbf{x}_i, y_i) in any way. For example, if via cross-validation one chooses the order of a polynomial and then fits via least squares to produce $\hat{f}_m(\mathbf{x})$, one must N times choose the order of polynomial and fit via least squares based on $\mathbf{T} - \{(\mathbf{x}_i, y_i)\}$, not simply refit coefficients for a polynomial of fixed order.

Stacking/Super-Learners

Honest K-fold Cross-Validation

The fundamental premise of cross-validation is that **whatever is going to be done to make final predictions must be done separately K times, once for each remainder from a fold**. Scrupulous adherence to this principle is all that stands between an analyst and completely unreliable (and usually overly optimistic) projections of performance.

If, for example, $\hat{f}_1(\mathbf{x}), \hat{f}_2(\mathbf{x}), \dots, \hat{f}_M(\mathbf{x})$ are combined through a random forest, the "OOB error" developed in production of that forest is no good indicator of Err for the super-learner (2). That is because although case i vector of predictions $(\hat{f}_1(\mathbf{x}_i), \hat{f}_2(\mathbf{x}_i), \dots, \hat{f}_M(\mathbf{x}_i))$ may seem to be "OOB" for a given tree, case i is used in the predictions for other cases and is thus used in the random forest meta-prediction of case i .

Stacking/Super-Learners

Honest K-fold Cross-Validation Continued

Again, honest cross-validation—without which reliable assessment of the likely performance of a choice of parameters for $\hat{f}_1, \hat{f}_2, \dots, \hat{f}_M$ and a top-level form \hat{f} (with an eye to ultimately good choice of these parameters)—is impossible, requires that

1. a training set must be split into K folds and
2. **all** must be redone separately K times on remainders, and used to predict folds

in order to develop sound projections for Err . This implies a large computational load (especially if repeated cross-validation is done) in order to choose an effective final version of super-learner for application.

A first (and best if computational burden were no concern) version of this is that where associated with each $\hat{f}_m(\mathbf{x})$ and also with the top-level form \hat{f} are grids of possible values of complexity parameters and a huge **product grid** is searched for a best cross-validation error and ultimately the optimizing set of parameters is applied to make the meta-predictor (2).

A second version of this application of honest cross-validation

pertains where individually-optimized versions of the \hat{f}_m will ultimately be combined into a form (2) and choice of complexity parameters for only the top-level form \hat{f} is attempted. **Notice** that when algorithms represented by the \hat{f}_m are themselves tuned (by K -fold cross-validation across some grids of parameter values) the resulting parameters need NOT be good ones for use in an ensemble. But they can be taken as given and search across a grid of complexity parameters for the top level form can be done using cross-validation errors produced according to the outline on panel 14 for each set of top-level parameters.

It is clear then that computation grows rapidly with the complexity of constituent predictor forms, the breath of the optimization desired, and the extent to which repetition of cross-validation is used.

Stacking/Super-Learners

What Form of Super-Learner?

What kind of top-level \hat{f} should be used in predictor (2) can be investigated by comparison of cross-validation errors. The linear form (1) is most common and (at least in its ad hoc application) famously successful. There is a good case to be made that a random forest form has potential to be at least as effective in this role. Its invariance to scale of predictors (inherited from its tree-based heritage) and wide success and reputation as an all-purpose tool make it a natural candidate.