Stat 648 Outline*

Steve Vardeman
Iowa State University
May 10, 2009

Abstract
This outline summarizes the main points of lectures based on the book
The Elements of Statistical Learning by Hastie, Tibshirani, and Friedman.
Some of the later material here benefits from Izenman’s Modern Multivariate Statistical Techniques.

Contents
1 Introduction (Ch 1 and Ch 2 of HTF) 4
  1.1 Some Very Simple Probability Theory and Decision Theory and Prediction Rules .................. 5
  1.2 Two Conceptually Simple Ways One Might Approximate a Conditional Mean .................. 6
      1.2.1 Nearest Neighbor Rules .................. 6
      1.2.2 Rules Based on a Linear Assumption About the Form of the Conditional Mean .................. 6
  1.3 The Curse of Dimensionality .................. 7
  1.4 Variance-Bias Considerations in Prediction .................. 8
  1.5 Some Fancier Ways One Might Approximate a Conditional Mean 9
      1.5.1 (Non-)Linear Regression Methods (Least Squares Fitting) 9
      1.5.2 Methods Based on Local Smoothing/Averaging .................. 10
  1.6 Complexity, Fit, and Choice of Tuning Parameters .................. 11

2 Linear Methods (Ch 3 of HTF) 11
  2.1 The Gram-Schmidt Process and the QR Decomposition .................. 12
  2.2 The Singular Value Decomposition of a Full Rank X .................. 14
  2.3 Ridge Regression and the Lasso and Some Generalizations Thereof 16
      2.3.1 Ridge Regression ............................ 16
      2.3.2 The Lasso and Generalizations .................. 17
  2.4 Two Methods With Derived Input Variables .................. 19

*Course development supported by NSF grant DMS #0502347 EMSW21-RTG awarded to the Department of Statistics, Iowa State University.
2.4.1 Principal Components Regression .......................... 19
2.4.2 Partial Least Squares Regression ......................... 20

3 Linear (and a Bit on Quadratic) Methods of Classification (Ch 4 of HTF) 22
3.1 Linear (and a bit on Quadratic) Discriminant Analysis ........ 22
3.2 Logistic Regression ......................................... 25
3.3 Separating Hyperplanes ..................................... 26

4 Basis Expansions and Regularization (Ch 5 of HTF) 27
4.1 Piecewise Polynomials and Regression Splines ................. 28
4.2 Smoothing Splines ........................................... 29
4.3 Generalizations to $p$-Dimensional Inputs ..................... 34
  4.3.1 Multi-Dimensional Regression Splines (Tensor Bases) .... 34
  4.3.2 Multi-Dimensional Smoothing Splines .................... 34
4.4 Reproducing Kernel Hilbert Spaces and Fitting Functions in $\mathbb{R}^p$ .... 35
  4.4.1 Some Special Cases ...................................... 39
4.5 Wavelet Bases .............................................. 40

5 Kernel Smoothing Methods (Ch 6 of HTF) 43
5.1 One-dimensional Kernel Smoothers ............................. 43
5.2 Kernel Smoothing in $p$ Dimensions ........................... 45
  5.2.1 Structured Kernels .................................... 45
  5.2.2 Structured Regression Functions ......................... 46
5.3 Kernel Density Estimation .................................... 47

6 Understanding and Predicting Predictor Performance (Ch 7 of HTF) 48
6.1 More on Predictor Squared Bias ................................ 49
6.2 Optimism of the Training Error ............................... 50
6.3 $C_p$, AIC and BIC ......................................... 51
  6.3.1 $C_p$ and AIC ......................................... 51
  6.3.2 BIC .................................................. 52
6.4 Cross-Validation Estimation of Err ............................ 53
6.5 Bootstrap Estimation of Err ................................... 55

7 Inference and Model Averaging (Ch 8 of HTF) 56
7.1 Bayes Predictors ............................................. 56
7.2 Predictors Built on Bootstrap Samples ......................... 58
  7.2.1 Bagging ............................................. 59
  7.2.2 Bumping ............................................. 59
7.3 Model Averaging and Stacking ................................. 59
  7.3.1 Bayesian Model Averaging ............................... 60
  7.3.2 Stacking ............................................. 61
8 Additive Models, Trees, and Related Methods (Ch 9 of HTF)  62
8.1 Additive Models .................................................. 62
8.2 Tree-Based Methods ............................................. 63
8.3 PRIM (Patient Rule Induction Method) ....................... 66
8.4 MARS (Multivariate Adapative Regression Splines) ....... 67

9 Boosting (Ch10 of HTF)  69
9.1 The AdaBoost Algorithm ........................................ 69
9.2 Why AdaBoost Might Be Expected to Work ............... 70
  9.2.1 Expected "Exponential Loss" for a Function of \( \mathbf{X} \) ... 70
  9.2.2 Sequential Search for a \( g(\mathbf{X}) \) With Small Expected Ex-
  ponential Loss and the AdaBoost Algorithm ............... 71
9.3 Other Forms of Boosting ....................................... 73
  9.3.1 SEL .......................................................... 74
  9.3.2 AEL (and Binary Regression Trees) ..................... 74
  9.3.3 \( K \)-Class Classification ................................. 75
9.4 Issues (More or Less Related to Boosting and Use of Trees as
  Base Predictors) ................................................ 76

10 Neural Networks (Ch 11 of HTF)  78
10.1 Projection Pursuit Regression ................................. 78
10.2 Neural Networks ................................................ 79
  10.2.1 The Back-propagation Algorithm ....................... 80
  10.2.2 Formal Regularization of Fitting ....................... 82

11 Support Vector Machines (Ch 12 of HTF)  84
11.1 The Linearly Separable Case ................................. 84
11.2 The Linearly Non-separable Case ............................ 88
11.3 SVM’s and Kernels ............................................ 89
  11.3.1 Heuristics .................................................. 90
  11.3.2 An Optimality Argument ............................... 91
11.4 Other Support Vector Stuff ................................. 93

12 Prototype Methods and Nearest Neighbors (Ch 13 of HTF)  94
1 Introduction (Ch 1 and Ch 2 of HTF)

The course is (mostly) about "supervised" machine learning for statisticians. The fundamental issue here is prediction, usually with a fairly large number of predictors and a large number of cases upon which to build a predictor. For \( \mathbf{X} \in \mathbb{R}^p \)

an input vector (a vector of predictors) with \( j \)th coordinate \( X_j \) and

some output value (usually univariate, and when it’s not, we’ll indicate so with some form of bold face) the object is to use \( N \) observed \((\mathbf{X}, Y)\) pairs

\[
(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)
\]

called training data to create an effective prediction rule

\[
\hat{Y}(\mathbf{X}) = f(\mathbf{X})
\]

"Supervised" learning means that the training data include values of the output variable.

A primary difference between a "machine learning" point of view and that common in basic graduate statistics courses is an ambivalence toward making statements concerning parameters of any models used in the invention of prediction rules (including those that would describe the amount of variation in observables one can expect the model to generate). Standard statistics courses often conceive of those parameters as encoding scientific knowledge about some underlying data-generating mechanism or real-world phenomenon. Machine learners don’t seem to care much about these issues, but rather are concerned about "how good" a prediction rule is.

An output variable might take values in \( \mathbb{R} \) or in some finite set \( G = \{1, 2, \ldots, K\} \).

In this latter case, it is often convenient to replace a \( G \)-valued output with \( K \) indicator variables

\[
Y_j = I[Y^* = j], \ j = 1, 2, \ldots, K
\]

(of course) each taking real values. This replaces the original \( Y^* \) with the vector output

\[
\mathbf{Y} = \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_K \end{pmatrix}
\]

taking values in \( \mathbb{R}^K \).
1.1 Some Very Simple Probability Theory and Decision Theory and Prediction Rules

For purposes of appealing to general theory that can suggest sensible forms for prediction rules, suppose that the random pair \((X, Y)\) has (joint) distribution \(P\). (Until further notice all expectations refer to distributions and conditional distributions derived from \(P\).) Suppose further that \(L(\hat{y}, y) \geq 0\) is some loss function quantifying how unhappy I am with prediction \(\hat{y}\) when in fact \(Y = y\). I might consider trying to choose the form of \(f\) to minimize

\[
EL(f(X), Y)
\]

This is (in theory) easily done, since

\[
EL(f(X), Y) = EE[L(f(X), Y) | X]
\]

and the minimization can be accomplished by minimizing the conditional mean for each possible value for \(X\). That is, an optimal choice of prediction rule is defined by

\[
f(x) = \arg \min_{a} E[L(a, Y) | X = x]
\] (1)

In the simple case where \(L(\hat{y}, y) = (\hat{y} - y)^2\), the optimal prediction rule (1) is well known to be

\[
f(x) = E[Y | X = x]
\]

In a simple two-class classification context, where \(Y^*\) takes values in \(\mathcal{G} = \{1, 2\}\), I might set up a decision problem with action space \(A = \mathcal{G}\) and loss function

\[
L(a, Y^*) = I[a \neq Y^*]
= I[a = 1] I[Y^* = 2] + I[a = 2] I[Y^* = 1]
\]

An optimal decision rule corresponding to (1) is

\[
a(x) = \begin{cases} 
1 & \text{if } E[I[Y^* = 1] | X = x] \geq .5 \\
2 & \text{if } E[I[Y^* = 1] | X = x] < .5 
\end{cases}
\]

Further, if I define \(Y = I[Y^* = 1]\) this is is

\[
a(x) = \begin{cases} 
1 & \text{if } E[Y | X = x] \geq .5 \\
2 & \text{if } E[Y | X = x] < .5 
\end{cases}
\]

So in both the squared error loss context and in the 0-1 loss 2-state classification context, \((P)\) optimal choice of predictor involves the conditional mean of an output variable (albeit the second is a 0-1 output variable). These strongly suggest the importance of finding ways to approximate \(E[Y | X = x]\) in order to (either directly or indirectly) produce good prediction rules.
1.2 Two Conceptually Simple Ways One Might Approximate a Conditional Mean

HTF introduce two standard and conceptually appealing ways of approximating $E[Y | X = x]$ (for $(X, Y)$ with distribution $P$). The tacit assumption here is that the training data are realizations of $(X_1, Y_1), (X_2, Y_2), \ldots, (X_N, Y_N)$ iid $P$.

1.2.1 Nearest Neighbor Rules

One idea is to think that if $N$ is "big enough," perhaps something like

$$E[Y | X = x] \approx \frac{1}{\# \text{ of } x_i = x} \sum_{i \text{ with } x_i = x} y_i$$

might work as an approximation to $E[Y | X = x]$. But almost always (unless $N$ is huge and the distribution of $X$ is discrete) the number of $x_i = x$ is only 0 or 1. So some modification is surely required. Perhaps the condition

$$x_i = x$$

might be replaced with

$$x_i \approx x$$

in (2).

One form of this is to define for each $x$ the $k$-neighborhood

$$N_k(x) = \text{the set of } k \text{ inputs } x_i \text{ in the training set closest to } x \text{ in } \mathbb{R}^p$$

A $k$-nearest neighbor approximation to $E[Y | X = x]$ is then

$$m_k(x) = \frac{1}{k} \sum_{i \text{ with } x_i \in N_k(x)} y_i$$

suggesting the prediction rule

$$\hat{Y}(X) = m_k(X)$$

One might hope that upon allowing $k$ to increase with $N$, provided that $P$ is not so bizarre this could be an effective predictor.

1.2.2 Rules Based on a Linear Assumption About the Form of the Conditional Mean

$k$-nearest neighbor prediction rules potentially provide very flexible forms, but also potentially require huge $N$ to produce reliable results. Something far simpler, far less flexible, but (when appropriate) also more reliably fit to a training data set is a linear form

$$l(x) = \beta'x$$
Letting

\[ \mathbf{X} = \begin{pmatrix} \mathbf{x}_1' \\ \mathbf{x}_2' \\ \vdots \\ \mathbf{x}_N' \end{pmatrix} \]

be a matrix with rows the input vectors for the \( N \) cases in the training data and

\[ \mathbf{Y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} \]

the corresponding vector of output values in the training set, the usual Stat 511 material says that least squares fitting of a linear form of conditional mean to the training data can be accomplished as

\[ \hat{\mathbf{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y} \]

with corresponding predictor

\[ \hat{Y}(\mathbf{X}) = \hat{\mathbf{\beta}}'\mathbf{X} \]

Regarding notation: As we must repeatedly refer to both the vector of inputs and the matrix of training values for the inputs, and considerable confusion is possible if we use the same bold face for both, I have decided to adopt the this dual notation of \( \mathbf{X} \) for the first and \( \mathbf{X} \) for the second. Since it is rare that one wants to refer to vector outputs, the notation \( \mathbf{Y} \) will be mostly unambiguous as the (column) vector of output values in the training data.

### 1.3 The Curse of Dimensionality

Roughly speaking, it’s clear that one wants to allow the most flexible kinds of prediction rules that a given training data set will adequately support fitting. Naively, one might hope that with what seem like the "large" sample sizes in machine learning contexts, even if \( p \) is fairly large, highly flexible prediction rules (like \( k \)-nearest neighbor rules) might be practically useful (as points "fill up" \( \mathbb{R}^p \) and allow one to figure out how \( \mathbf{Y} \) behaves as a function of \( \mathbf{X} \), operating quite locally). But our intuition about how sparse data sets in \( \mathbb{R}^p \) actually must be is pretty weak and needs some education. There is the "curse of dimensionality" to be reckoned with.

There are many ways of framing this inescapable sparsity. Some simple ones involve facts about uniform distributions on the unit ball in \( \mathbb{R}^p \)

\[ \{ \mathbf{x} \in \mathbb{R}^p | \|\mathbf{x}\| \leq 1 \} \]

and on a unit cube centered at the origin

\[ [-.5, .5]^p \]
For one thing, "most" of these distributions are very near the surface of the solids. The cube \([-r, r]^p\) capturing (for example) half the volume of the cube (half the probability mass of the distribution) has

\[ r = .5 (.5)^{1/p} \]

which converges to .5 as \(p\) increases. Essentially the same story holds for the uniform distribution on the unit ball, except that the radius capturing half the probability mass has

\[ r = (.5)^{1/p} \]

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

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

\[ R = \text{the distance from the origin to a single random point} \]

\(R\) has cdf

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

So if \(R_1, R_2, \ldots, R_N\) are iid with this distribution, \(M = \min\{R_1, R_2, \ldots, R_N\}\) has cdf

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

This distribution has, for example, median

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

For, say, \(p = 100\) and \(N = 10^6\), the median of the distribution if \(M\) is .87.

In addition to these kinds of considerations of sparsity, there is the fact that the potential complexity of functions of \(p\) variables explodes exponentially in \(p\).

### 1.4 Variance-Bias Considerations in Prediction

Just as in model parameter estimation, the "variance-bias trade-off" notion is important in a prediction context. Here we consider a decomposition of what HTF in their Section 2.9 call the "test/generalization error" that can help us think about that trade-off. In what follows, assume that \((X_1, Y_1), (X_2, Y_2), \ldots, (X_N, Y_N)\) are iid \(P\) and independent of \((X, Y)\) that is also \(P\) distributed. Write \(E_N\) for expectation with respect to the joint distribution of the training data
and \( E[|X = x] \) for conditional expectation (and \( \text{Var}[Y : X = x] \) for conditional variance) based on the conditional distribution of \( Y|X = x \). For a fixed \( x \), suppose that \( f(x) \) is a function of \((X_1, Y_1), (X_2, Y_2), \ldots, (X_N, Y_N)\). A measure of the effectiveness of the prediction \( f(x) \) for \( Y \) at \( X = x \) is

\[
E_N \left[ (f(x) - Y)^2 | X = x \right]
\]

\[
= E_N \left\{ (f(x) - E[Y|X = x])^2 + E[(Y - E[Y|X = x])^2 | X = x] \right\}
\]

\[
= E_N \left\{ (f(x) - E_N f(x))^2 + (E_N f(x) - E[Y|X = x])^2 \right\} + \text{Var}[Y|X = x]
\]

\[
= \text{Var}_N (f(x)) + (E_N f(x) - E[Y|X = x])^2 + \text{Var}[Y|X = x]
\]

The first quantity, \( \text{Var}_N (f(x)) \), is the variance of the prediction at \( x \). The second term, \( (E_N f(x) - E[Y|X = x])^2 \), is a kind of "bias" of the prediction at \( x \). And \( \text{Var}[Y|X = x] \) is an unavoidable variance in output at \( X = x \).

1.5 Some Fancier Ways One Might Approximate a Conditional Mean

There are many important tools for producing prediction rules/approximating conditional means. HTF sketch a number of them at the end of their Chapter 2. In some ways, listed are methods that fit between the extremes (in terms of flexibility/complexity of a prediction rule) of least squares fitting of a linear form and nearest neighbor rules.

1.5.1 (Non-)Linear Regression Methods (Least Squares Fitting)

One basic form of prediction rule is

\[
\hat{Y}(x) = \sum_{m=1}^{M} \theta_m h_m(x)
\]

for some set of "basis" functions on \( \mathbb{R}^p \), \( \{h_m(x)\} \). These basis functions \( h_m(x) \) might be

1. powers and products of powers of coordinates of \( x \). The result is polynomial regression.
2. sines and cosines of known multiples of coordinates of \( x \). The result is Fourier analysis.
3. of the form

\[
\frac{1}{1 + \exp(\alpha_m + \beta_m^T x)}
\]

where the \( \alpha_m \) and \( \beta_m \) are unknown parameters that are included in the fitting. The result is a kind of "neural network" and fitting the parameters is a non-linear regression problem.
4. "spline functions." For example in $p = 1$ dimension with "knots" $t_1 < t_2 < \cdots < t_{M-2}$, one might have

\[ h_m (x) = (x - t_m)_+ \text{ for } m = 1, 2, \ldots, M - 2 \]
\[ h_{M-1} (x) = x \text{ and } h_M (x) = 1 \]

Here predictions are piecewise linear in $x$. Products of such things with arguments coordinates of $x \in \mathbb{R}^p$ can be used for $p > 1$. Where knots are fixed, fitting is linear regression problem. If choosing knots is part of the fitting process, this is a non-linear regression problem.

5. "radial basis functions" (radially symmetric functions) like

\[ \exp \left( -\frac{1}{2\lambda_m} \| x - \mu_m \|^2 \right) \]

for some parameters $\lambda_m$ and $\mu_m$. Where these are fixed, fitting is linear regression problem. If choosing the $\lambda_m$ and $\mu_m$ part of the fitting process, this is a non-linear regression problem.

1.5.2 Methods Based on Local Smoothing/Averaging

One might try choosing a function $f(x)$ by balancing some measure of "fit" against a penalty associated with how "wiggly/bizarre" the function is. For $p = 1$, it is possible to solve for $\lambda > 0$ the optimization problem

\[ f_\lambda = \arg \min_{h \text{ with enough derivatives}} \left( \sum_{i=1}^{N} (y_i - h (x_i))^2 + \lambda \int |h'' (x)|^2 \, dx \right) \]

and the optimizing function is a so-called "cubic smoothing spline." The integral above might be called $J(h)$, and the larger is $\lambda$, the less wiggly is $f_\lambda$. For $p > 1$, one might try considering fitting functions of the form

\[ h (x) = \sum_{j=1}^{p} h_j (x_j) \]

using a penalty like

\[ \lambda \sum_{j=1}^{p} J(h_j) \]

and hope to do something generalizing the $p = 1$ method.

Similarly, one might consider a "projection pursuit regression" model that fits

\[ \hat{Y} (x) = \sum_{m=1}^{M} g_m (\beta'_m x) \]
where the $\beta_m$ are unknown parameters and the real-valued functions of a single real variable $g_m$ are to be chosen as well (presumably using some penalty function idea and an iterative scheme that alternatively chooses $\beta_m$’s and $g_m$’s).

Another possibility is the local use of weighted regression ideas. For some kernel function $\kappa_\lambda (x_0, x)$, like for example,

$$\kappa_\lambda (x_0, x) = \frac{1}{\lambda} \exp \left(-\frac{1}{2\lambda} \|x_0 - x\|^2\right)$$

and some (usually fairly simple) function like

$$h_\beta (x) = \beta \in \mathbb{R} \text{ or } h_\beta (x) = \beta' x$$

one might choose for each $x_0$

$$\beta (x_0) = \arg \min_{\beta} \sum_{i=1}^N \kappa_\lambda (x_0, x_i) (y_i - h_\beta (x_i))^2$$

and then employ the prediction rule

$$\hat{Y} (x) = h_{\beta(x)} (x)$$

(This produces smoothers like the loess smoother.)

1.6 Complexity, Fit, and Choice of Tuning Parameters

The methods presented at the end of Chapter 2 all have complexity parameters (the number of basis functions employed, the penalty weight, and the band width/neighborhood size parameter) that must be chosen by a user. If a choice of complexity parameter doesn’t allow enough flexibility in the form of a predictor, under-fit occurs and there is large bias in prediction. On the other hand, if the choice allows too much flexibility, bias may be reduced, but the price paid is large variance and over-fit. It is a theme that runs through this material that complexity must be chosen in a way that balances variance and bias for the particular combination of $N$ and $p$ and general circumstance one faces. Specifics of methods for doing this are presented later in the book.

2 Linear Methods (Ch 3 of HTF)

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

$$\hat{Y} (X) = \beta' X$$

for an appropriate $\beta \in \mathbb{R}^p$ than what is said in Stat 500 and Stat 511 (where least squares is used to fit the linear form to all $p$ input variables or to some subset of $M$ of them). Some of that involves more linear algebra background than is assumed or used in Stat 511. We begin with some linear theory that
is useful both here and later, and then go on to consider other fitting methods besides ordinary least squares.

Consider again the matrix of training data inputs

$$
\mathbf{X} = \begin{pmatrix}
x'_1 \\
x'_2 \\
\vdots \\
x'_N
\end{pmatrix}
$$

and corresponding vector of training data outputs

$$
\mathbf{Y} = \begin{pmatrix}
y_1 \\
y_2 \\
\vdots \\
y_N
\end{pmatrix}
$$

It is standard Stat 511 fare that least squares projects \( \mathbf{Y} \) onto \( \mathcal{C}(\mathbf{X}) \), the column space of \( \mathbf{X} \), in order to produce the vector of fitted values

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

2.1 The Gram-Schmidt Process and the QR Decomposition

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

$$
\langle \mathbf{u}, \mathbf{v} \rangle = \sum_{i=1}^{N} u_i v_i = \mathbf{u}' \mathbf{v}
$$

and describe the process in these terms.

In what follows, vectors are \( N \)-vectors and in particular, \( x_j \) is the \( j \)th column of \( \mathbf{X} \). Notice that this is in potential conflict with earlier notation that made \( x_i \) the \( p \)-vector of inputs for the \( i \)th case in the training data. We will simply have to read the following in context and keep in mind the local convention. With this notation, the Gram-Schmidt process proceeds as follows:
1. Set the first column of $Z$ to be 
   \[ z_1 = x_1 \]

2. Having constructed $Z_{l-1} = (z_1, z_2, \ldots, z_{l-1})$, let 
   \[ z_l = x_l - \sum_{j=1}^{l-1} \left< x_l, z_j \right> z_j \]

   It is easy enough to see that $\left< z_l, z_j \right> = 0$ for all $j < l$ (building up the orthogonality of $z_1, z_2, \ldots, z_{l-1}$ by induction), since 
   \[ \left< z_l, z_j \right> = \left< x_l, z_j \right> - \left< x_l, z_j \right> \]
   as at most one term of the sum in step 2. above is non-zero. Further, assume that $C(Z_{l-1}) = C(X_{l-1})$. Since $z_l \in C(X_l)$ and so $C(Z_l) \subset C(X_l)$. And since any element of $C(X_l)$ can be written as a linear combination of an element of $C(X_{l-1}) = C(Z_{l-1})$ and $x_l$, we also have $C(X_l) \subset C(Z_l)$. Thus, as advertised, $C(Z_l) = C(X_l)$.

   Since the $z_j$ are perpendicular, for any $N$-vector $w$,
   \[ \sum_{j=1}^{l} \left< w, z_j \right> \]
   is the projection of $w$ onto $C(Z_l) = C(X_l)$. (To see this, consider minimization of the quantity $\left< w - \sum_{j=1}^{l} c_j z_j, w - \sum_{j=1}^{l} c_j z_j \right> = \left< w, w \right> - \sum_{j=1}^{l} c_j z_j$ by choice of the constants $c_j$.) In particular, the sum on the right side of the equality in step 2. of the Gram-Schmidt process is the projection of $x_l$ onto $C(X_{l-1})$. And the projection of $Y$ onto $C(X_l)$ is
   \[ \sum_{j=1}^{l} \left< Y, z_j \right> \]

   This means that for a full $p$-variable regression,
   \[ \frac{\left< Y, z_p \right>}{\left< z_p, z_p \right>} \]
   is the regression coefficient for $z_p$ and (since only $z_p$ involves it) the last variable in $X$, $x_p$. So, in constructing a vector of fitted values, fitted regression coefficients in multiple regression can be interpreted as weights to be applied to that part of the input vector that remains after projecting onto the space spanned by all the others.

   The construction of the orthogonal variables $z_j$ can be represented in matrix form as
   \[ X_{N \times p} = Z_{N \times p} \Gamma_{p \times p} \]
where $\Gamma$ is upper triangular with
\[
\gamma_{kj} = \langle z_k, x_j \rangle \quad \text{the value in the } k\text{th row and } j\text{th column of } \Gamma
\]
\[
= \frac{\langle z_k, z_j \rangle}{\langle z_k, z_k \rangle}
\]

Defining
\[
D = \text{diag}\left(\langle z_1, z_1 \rangle^{1/2}, \ldots, \langle z_p, z_p \rangle^{1/2}\right) = \text{diag}\left(\|z_1\|, \ldots, \|z_p\|\right)
\]
and letting
\[
Q = ZD^{-1} \quad \text{and} \quad R = D\Gamma
\]
one may write
\[
X = QR
\]
that is the so-called $QR$ decomposition of $X$.

In (3), $Q$ is $N \times p$ with
\[
Q'Q = D'Z'ZD = D'\text{diag}\left(\langle z_1, z_1 \rangle, \ldots, \langle z_p, z_p \rangle\right)D = I
\]
That is, $Q$ has for columns perpendicular unit vectors that form a basis for $C(X)$. $R$ is upper triangular and that says that only the first $l$ of these unit vectors are needed to create $x_l$.

The decomposition is computationally useful in that (for $q_j$ the $j$th column of $Q$) the projection of $Y$ onto $C(X)$ is (since the $q_j$ comprise an orthonormal basis for $C(X)$)
\[
\hat{Y} = \sum_{j=1}^{p} \langle Y, q_j \rangle q_j = QQ'Y
\]
and
\[
\hat{\beta}^{\text{OLS}} = R^{-1}Q'Y
\]
(The fact that $R$ is upper triangular implies that there are efficient ways to compute its inverse.)

### 2.2 The Singular Value Decomposition of a Full Rank $X$

The matrix $X$ has a so-called singular value decomposition as
\[
X_{N \times p} = U_{N \times p} D_{p \times p} V_{p \times p}'
\]
where $U$ has orthonormal columns spanning $C(X)$, $V$ has orthonormal columns spanning $C(X')$ (the row space of $X$), and $D = \text{diag}(d_1, d_2, \ldots, d_p)$ for
\[
d_1 \geq d_2 \geq \cdots \geq d_p > 0
\]
The $d_j$ are the "singular values" of $X$. 

14
An interesting property of the singular value decomposition is this. If $U_l, V_l,$ and $D_l$ are matrices consisting of the first $l$ columns of $U, V,$ and $D$ respectively, then

$$X_l = U_l D_l V_l'$$

is the best (in the sense of squared distance from $X$ in $\mathbb{R}^{Np}$) rank-$l$ approximation to $X$.

Since the columns of $U$ are an orthonormal basis for $C(X)$, the projection of $Y$ onto $C(X)$ is

$$\hat{Y}_{OLS} = \sum_{j=1}^{p} \langle Y, u_j \rangle u_j = UU'Y$$

This is, of course, completely parallel to (4) and is a consequence of the fact that the columns of both $U$ and $Q$ (from the QR decomposition of $X$) form orthonormal bases for $C(X)$. In general, the two bases are not the same.

Now using the singular value decomposition of $X$

$$X'X = VD'U'UDV' = VD^2V'$$

which is the "eigen decomposition" of the symmetric and positive definite $X'X$. The eigenvectors $v_1, v_2, \ldots, v_p$ (the columns of $V$) are called the "principal components directions" of $X$ in $\mathbb{R}^{p}$. The vector

$$z_1 = Xv_1$$

is the product $Xw$ with the largest sample variance of entries (or largest squared length in $\mathbb{R}^N$) subject to the constraint that $\|w\| = 1$. This sample variance is

$$\frac{1}{N} z_1'z_1 = \frac{1}{N} v_1'X'Xv_1 = \frac{1}{N} v_1'VD^2V'v_1 = \frac{d_1^2}{N}$$

A second representation of $z_1$ is

$$z_1 = Xv_1 = UDv_1 = UD \left( \begin{array}{c} 1 \\ 0 \\ \vdots \\ 0 \end{array} \right) = d_1 u_1$$

and this vector is called the "first principal component of $X$" (in $\mathbb{R}^N$).

In general, the $j$th principal component of $X$ is $z_j = Xv_j = d_j u_j$, which is the vector of the form $Xw$ with the largest sample variance of entries subject to the constraints that $\|w\| = 1$ and $\langle z_j, z_l \rangle = 0$ for all $l < j$. This sample variance is $\frac{1}{N} z_j'z_j = d_j^2/N$.

Notice that when $X$ has standardized columns (i.e. each column of $X$, $x_j$, has $\langle 1, x_j \rangle = 0$ and $\langle x_j, x_j \rangle = N$), the $p \times p$ matrix

$$\frac{1}{N} X'X$$
is the sample covariance (and correlation) matrix for the $p$ input variables $X_1, X_2, \ldots, X_p$.

### 2.3 Ridge Regression and the Lasso and Some Generalizations Thereof

An alternative to seeking to find a suitable level of complexity in a linear prediction rule through subset selection and least squares fitting of a linear form to the selected variables, is to employ a shrinkage method based on a penalized version of least squares to choose a vector $\beta \in \mathbb{R}^p$ to employ in a linear prediction rule. Here we consider several such methods. The implementation of these methods is not equivariant to the scaling used to express the input variables $X_j$. So that we can talk about properties of the methods that are associated with a well-defined scaling, we assume here that the output variable has been centered (i.e. that $\langle Y, 1 \rangle = 0$) and that the columns of $X$ have been standardized (and if originally $X$ had a constant column, it has been removed).

#### 2.3.1 Ridge Regression

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

$$
\hat{\beta}_\lambda^\text{ridge} = \arg \min_{\beta \in \mathbb{R}^p} \left\{ \sum_{i=1}^{N} \left( y_i - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\} \quad (6)
$$

Here $\lambda$ is a penalty/complexity parameter that controls how much $\hat{\beta}_\text{OLS}$ is shrunk towards $0$. The unconstrained minimization problem expressed in (6) has an equivalent constrained minimization description as

$$
\hat{\beta}_t^\text{ridge} = \arg \min_{\beta \text{ with } \sum_{j=1}^{p} \beta_j^2 \leq t} \sum_{i=1}^{N} \left( y_i - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \quad (7)
$$

for an appropriate $t > 0$. (Corresponding to $\lambda$ used in (6), is $t = \|\hat{\beta}_\lambda^\text{ridge}\|^2$ used in (7). Conversely, corresponding to $t$ used in (7), one may use a value of $\lambda$ in (6) producing the same error sum of squares.)

The unconstrained form (6) calls upon one to minimize

$$
(Y - X\beta)' (Y - X\beta) + \lambda \beta' \beta
$$

and some vector calculus leads directly to

$$
\hat{\beta}_\lambda^\text{ridge} = (X'X + \lambda I)^{-1} X' Y
$$
So then, using the singular value decomposition of $X$, 

$$
\hat{Y}^{\text{ridge}} = X\hat{\beta}_\lambda^{\text{ridge}} \\
= UD V' (VDU'UDV' + \lambda I)^{-1} VDU' Y \\
= UD (V' (VDU'UDV' + \lambda I) V)^{-1} D U' Y \\
= UD (D^2 + \lambda I)^{-1} DU' Y \\
= \sum_{j=1}^{p} \left( \frac{d_j^2}{d_j^2 + \lambda} \right) \langle Y, u_j \rangle u_j 
$$

Comparing to (5) and recognizing that 

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

we see that the coefficients of the orthonormal basis vectors $u_j$ employed to get $\hat{Y}^{\text{ridge}}$ are shrunken versions of the coefficients applied to get $\hat{Y}^{\text{OLS}}$. The most severe shrinking is enforced in the directions of the smallest principal components of $X$ (the $u_j$ least important in making up low rank approximations to $X$).

The function 

$$
df(\lambda) = \text{tr} \left( (X'X + \lambda I)^{-1} X' \right) \\
= \text{tr} \left( UD (D^2 + \lambda I)^{-1} D U' \right) \\
= \text{tr} \left( \sum_{j=1}^{p} \left( \frac{d_j^2}{d_j^2 + \lambda} \right) u_j u_j' \right) \\
= \text{tr} \left( \sum_{j=1}^{p} \left( \frac{d_j^2}{d_j^2 + \lambda} \right) u_j' u_j \right) \\
= \sum_{j=1}^{p} \left( \frac{d_j^2}{d_j^2 + \lambda} \right)
$$

is called the effective degrees of freedom associated with the ridge regression. (Note that if $\lambda = 0$ ridge regression is ordinary least squares and this is $p$, the usual degrees of freedom associated with projection onto $C(X)$, i.e. trace of the projection matrix onto this column space. As $\lambda \to \infty$, this goes to 0.)

2.3.2 The Lasso and Generalizations

The "lasso" and the other relatives of ridge regression are the result of generalizing the optimization criteria (6) and (7) by replacing $\sum_{j=1}^{p} \beta_j^2$ with $\sum_{j=1}^{p} |\beta_j|^q$
for a $q > 0$. That produces

$$\hat{\beta}_q = \arg \min_{\beta \in \mathbb{R}^p} \left\{ \sum_{i=1}^N \left( y_i - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p |\beta_j|^q \right\}$$  \hspace{1cm} (9)

generalizing (6) and

$$\hat{\beta}_q^t = \arg \min_{\beta \text{ with } \sum_{j=1}^p |\beta_j|^t \leq t} \sum_{i=1}^N \left( y_i - \sum_{j=1}^p \beta_j x_{ij} \right)^2$$  \hspace{1cm} (10)

generalizing (7). The so called "lasso" is the $q = 1$ case of (9) or (10). That is, for $t > 0$

$$\hat{\beta}_t^{\text{lasso}} = \arg \min_{\beta \text{ with } \sum_{j=1}^p |\beta_j| \leq t} \sum_{i=1}^N \left( y_i - \sum_{j=1}^p \beta_j x_{ij} \right)^2$$  \hspace{1cm} (11)

Because of the shape of the constraint region

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

(in particular its sharp corners at coordinate axes) some coordinates of $\hat{\beta}_t^{\text{lasso}}$ are often 0, and the lasso automatically provides simultaneous shrinking of $\hat{\beta}_t^{\text{OLS}}$ toward 0 and rational subset selection. (The same is true of cases of (10) with $q < 1$.)

In Table 3.4, HTF provide explicit formulas for fitted coefficients for the case of $X$ with orthonormal columns. These translate in the present case of $X$ with standardized and orthogonal columns to the following.

<table>
<thead>
<tr>
<th>Method of Fitting</th>
<th>Fitted Coefficient for $X_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLS</td>
<td>$\hat{\beta}_j^{\text{OLS}}$</td>
</tr>
<tr>
<td>Best Subset (of Size $M$)</td>
<td>$\tilde{\beta}_j^{\text{OLS}}$ $I \left[ \text{rank} \left</td>
</tr>
<tr>
<td>Ridge Regression</td>
<td>$\tilde{\beta}_j^{\text{OLS}} \left( \frac{1}{1+NX} \right)$</td>
</tr>
<tr>
<td>Lasso</td>
<td>$\tilde{\beta}_j^{\text{OLS}} \left( \frac{1}{1+NX} \right) \left( \text{sign} \tilde{\beta}_j^{\text{OLS}} \right) \left(</td>
</tr>
</tbody>
</table>

Best subset regression provides a kind of "hard thresholding" of the least squares coefficients, ridge regression provides shrinking of all coefficients toward 0, and the lasso provides a kind of "soft thresholding" of the coefficients.

Chapter 3 of the 2nd Edition of HTF provides some modifications of the ridge/lasso idea. One is the "elastic net" idea. This is a kind of compromise
between the ridge and lasso. For an $\alpha \in (0, 1)$ and some $t > 0$, this is defined by

$$
\hat{\beta}_t^{\text{elastic net}} = \arg \min_{\beta} \text{ with } \sum_{j=1}^{p} (\alpha \beta_j^2 + (1-\alpha) |\beta_j|) \leq t \sum_{i=1}^{N} \left( y_i - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2
$$

The constraint is some kind of compromise between the ridge and lasso constraints. The constraint regions have "corners" like the lasso regions but are more rounded than the lasso regions.

### 2.4 Two Methods With Derived Input Variables

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

#### 2.4.1 Principal Components Regression

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

$$
z_j = X v_j = d_j u_j
$$

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

$$
\hat{Y}_{\text{PCR}} = \sum_{j=1}^{M} \langle Y, z_j \rangle z_j
$$

$$
= \sum_{j=1}^{M} \langle Y, u_j \rangle u_j
$$

Comparing this to (5) and (8) we see that ridge regression shrinks the coefficients of the principal components $u_j$ according to their importance in making up $X$, while principal components regression "zeros out" those least important in making up $X$.
Notice too, that \( \hat{Y}^{PCR} \) can be written in terms of the original inputs as

\[
\hat{Y}^{PCR} = \sum_{j=1}^{M} \langle Y, u_j \rangle \frac{1}{d_j} X v_j
\]

\[
= X \left( \sum_{j=1}^{M} \langle Y, u_j \rangle \frac{1}{d_j} v_j \right)
\]

\[
= X \left( \sum_{j=1}^{M} \frac{1}{d_j^2} \langle Y, X v_j \rangle v_j \right)
\]

so that

\[
\hat{\beta}^{PCR} = \sum_{j=1}^{M} \frac{1}{d_j^2} \langle Y, X v_j \rangle v_j
\]  

(13)

where (obviously) the sum above has only \( M \) terms. \( \hat{\beta}^{OLS} \) is the \( p = M \) version of (13). As the \( v_j \) are orthonormal, it is therefore clear that

\[
\| \hat{\beta}^{PCR} \| \leq \| \hat{\beta}^{OLS} \|
\]

So principal components regression shrinks both \( \hat{Y}^{PCR} \) toward 0 in \( \mathbb{R}^N \) and \( \hat{\beta}^{OLS} \) toward 0 in \( \mathbb{R}^p \).

### 2.4.2 Partial Least Squares Regression

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

The logic of partial least squares is this. Suppose that

\[
z_1 = X w_1
\]

is the linear combination of columns of \( X \) maximizing

\[
|\langle Y, X w \rangle| \quad (14)
\]

(which is essentially the sample covariance between the variables \( Y \) and \( X' w \)) subject to the constraint that \( \| w \| = 1 \). Then for \( j > 1 \), let

\[
z_j = X w_j
\]

be the linear combination of columns of \( X \) maximizing

\[
|\langle Y, X w \rangle| \quad (15)
\]
subject to the constraints that \( \| \mathbf{w} \| = 1 \) and \( \langle \mathbf{z}_j, \mathbf{z}_i \rangle = 0 \) for all \( l < j \). (Note that upon replacing \( \langle \mathbf{Y}, \mathbf{X} \mathbf{w} \rangle \) with \( \langle \mathbf{X} \mathbf{w}, \mathbf{X} \mathbf{w} \rangle \) one gets the principal components of \( \mathbf{X} \) in place of the PLS predictors.) Principal components regression uses the first \( M \) of these variables \( \mathbf{z}_j \) as input variables.

HTF are not terribly clear about an algorithm for accomplishing the computation of the PLS predictors. I believe that the PLS algorithm is as follows. Take

\[
\mathbf{z}_1 = \sum_{j=1}^{p} \langle \mathbf{Y}, \mathbf{x}_j \rangle \mathbf{x}_j
\]

and define

\[
\mathbf{X}^0 = \mathbf{X}
\]

For \( l \geq 1 \) define \( \mathbf{X}^l \) by orthogonalizing the columns of \( \mathbf{X}^{l-1} \) with respect to \( \mathbf{z}_l \). That is, define the \( j \)th column of \( \mathbf{X}^l \) by

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

Then take

\[
\mathbf{z}_{l+1} = \sum_{j=1}^{p} \langle \mathbf{Y}, \mathbf{x}_j^l \rangle \mathbf{x}_j^l
\]

So constructed, the PLS predictors \( \mathbf{z}_j \) are orthogonal. Using the first \( M \) of these as regressors, one has the vector of fitted output values

\[
\hat{\mathbf{Y}}_{\text{PLS}} = \sum_{j=1}^{M} \frac{\langle \mathbf{Y}, \mathbf{z}_j \rangle}{\langle \mathbf{z}_j, \mathbf{z}_j \rangle} \mathbf{z}_j
\]

Since the PLS predictors are (albeit recursively-computed data-dependent) linear combinations of columns of \( \mathbf{X} \), it is possible to find a \( p \)-vector \( \hat{\mathbf{\beta}}_{\text{PLS}}^M \) such that

\[
\hat{\mathbf{Y}}_{\text{PLS}} = \mathbf{X} \hat{\mathbf{\beta}}_{\text{PLS}}^M
\]

and thus produce a corresponding prediction rule

\[
\hat{\mathbf{Y}} (\mathbf{X}) = \left( \hat{\mathbf{\beta}}_{\text{PLS}}^M \right)' \mathbf{X}
\] (16)

It seems like in (16), the number of components, \( M \), should function as a complexity parameter. But then again there is the following. When the \( \mathbf{x}_j \) are orthogonal, it’s fairly easy to see that \( \mathbf{z}_1 \) is a multiple of \( \hat{\mathbf{Y}}_{\text{OLS}} \), \( \hat{\mathbf{\beta}}_1 = \hat{\mathbf{\beta}}_{\text{PLS}} = \hat{\mathbf{\beta}}_{\text{PLS}} = \cdots = \hat{\mathbf{\beta}}_{\text{PLS}} = \hat{\mathbf{\beta}}_{\text{PLS}} = \hat{\mathbf{\beta}}_{\text{PLS}} \), and all steps of partial least squares after the first are simply providing a basis for the orthogonal complement of the 1-dimensional subspace of \( C(\mathbf{X}) \) generated by \( \hat{\mathbf{Y}}_{\text{OLS}} \) (without improving fitting at all). That is, here changing \( M \) doesn’t change flexibility of the fit at all. (Presumably, when the \( \mathbf{x}_j \) are nearly orthogonal, something similar happens.)
3 Linear (and a Bit on Quadratic) Methods of Classification (Ch 4 of HTF)

Suppose now that an output variable \( Y^* \) takes values in \( \mathcal{G} = \{1, 2, \ldots, K\} \), or equivalently that \( Y \) is a \( K \)-variate set of indicators, \( Y_k = I [Y^* = k] \). The object here is to consider methods of producing prediction/classification rules \( \hat{Y}^* (X) \) taking values in \( \mathcal{G} \) (and mostly ones) that have sets \( \{ X \in \mathbb{R}^p | \hat{Y}^* (X) = k \} \) with boundaries that are defined (at least piece-wise) by linear equalities

\[
\beta' X = c
\]  

The most obvious/naive potential method here, namely to regress the \( K \) indicator variables \( Y_k \) onto \( X \) (producing least squares regression vector coefficients \( \hat{\beta}_k \)) and then to employ

\[
\hat{Y}^* (X) = \arg \max_k \hat{Y}_k (X) = \arg \max_k \hat{\beta}'_k X
\]

fails miserably because of the possibility of "masking" if \( K > 2 \). One must be smarter than this. Three kinds of smarter alternatives are Linear (and Quadratic) Discriminant Analysis, Logistic Regression, and direct searches for separating hyperplanes. The first two of these are "statistical" in origin with long histories in the field.

3.1 Linear (and a bit on Quadratic) Discriminant Analysis

Suppose that for \( (X, Y^*) \sim P \), \( \pi_k = P [Y^* = k] \) and the conditional distribution of \( X \) on \( \mathbb{R}^p \) given that \( Y^* = k \) is \( \text{MVN}_p (\mu_k, \Sigma) \), i.e. the conditional pdf is

\[
f_k (x) = (2\pi)^{-p/2} (\det \Sigma)^{-1/2} \exp \left( -\frac{1}{2} (x - \mu_k)' \Sigma^{-1} (x - \mu_k) \right)
\]

Then it follows that

\[
\ln \left( \frac{P [Y^* = k | X = x]}{P [Y^* = l | X = x]} \right) = \ln \left( \frac{\pi_k}{\pi_l} \right) - \frac{1}{2} \mu_k' \Sigma^{-1} \mu_k + \frac{1}{2} \mu_l' \Sigma^{-1} \mu_l + x' \Sigma^{-1} (\mu_k - \mu_l)
\]

so that an optimal predictor/decision rule is

\[
\hat{Y}^* (X) = \arg \max_k \left[ \ln (\pi_k) - \frac{1}{2} \mu_k' \Sigma^{-1} \mu_k + X' \Sigma^{-1} (\mu_k - \mu_l) \right]
\]

and boundaries between regions in \( \mathbb{R}^p \) where \( \hat{Y}^* (X) = k \) and \( \hat{Y}^* (X) = l \) are subsets of the sets

\[
\left\{ X \in \mathbb{R}^p | X' \Sigma^{-1} (\mu_k - \mu_l) = -\ln \left( \frac{\pi_k}{\pi_l} \right) + \frac{1}{2} \mu_k' \Sigma^{-1} \mu_k - \frac{1}{2} \mu_l' \Sigma^{-1} \mu_l \right\}
\]
i.e. are defined by equalities of the form (17).

This is dependent upon all \( K \) conditional normal distributions having the same covariance matrix, \( \Sigma \). In the event these are allowed to vary, conditional distribution \( k \) with covariance matrix \( \Sigma_k \), an optimal predictor/decision rule is

\[
\hat{Y}^*(X) = \arg \max_k \left[ \ln(\pi_k) - \frac{1}{2} \ln(\det(\Sigma_k)) - \frac{1}{2} (X - \mu_k)' \Sigma_k^{-1} (X - \mu_k) \right]
\]

and boundaries between regions in \( \mathbb{R}^p \) where \( \hat{Y}^*(X) = k \) and \( \hat{Y}^*(X) = l \) are subsets of the sets

\[
\{ X \in \mathbb{R}^p | \frac{1}{2} (X - \mu_k)' \Sigma_k^{-1} (X - \mu_k) - \frac{1}{2} (X - \mu_l)' \Sigma_l^{-1} (X - \mu_l) = -\ln \left( \frac{\pi_k}{\pi_l} \right) - \frac{1}{2} \ln(\det(\Sigma_k)) + \frac{1}{2} \ln(\det(\Sigma_l)) \}
\]

Unless \( \Sigma_k = \Sigma_l \) this kind of set is a quadratic surface in \( \mathbb{R}^p \), not a hyperplane. One gets (not linear, but) Quadratic Discriminant Analysis.

Of course, in order to use LDA or QDA, one must estimate the vectors \( \mu_k \) and the covariance matrix \( \Sigma_k \) or matrices \( \Sigma_k \) from the training data. Estimating \( K \) potentially different matrices \( \Sigma_k \) requires estimation of a very large number of parameters. So thinking about QDA versus LDA, one is again in the situation of needing to find the level of predictor complexity that a given data set will support. QDA is a more flexible/complex method than LDA, but using it in preference to LDA increases the likelihood of over-fit and poor prediction.

One idea that has been offered as a kind of continuous compromise between LDA and QDA is for \( \alpha \in (0, 1) \) to use

\[
\hat{\Sigma}_k (\alpha) = \alpha \hat{\Sigma}_k + (1 - \alpha) \hat{\Sigma}_{pooled}
\]

in place of \( \hat{\Sigma}_k \) in QDA. This kind of thinking even suggests as an estimate of a covariance matrix common across \( k \)

\[
\hat{\Sigma} (\gamma) = \gamma \hat{\Sigma}_{pooled} + (1 - \gamma) \sigma^2 I
\]

for \( \gamma \in (0, 1) \) and \( \sigma^2 \) an estimate of variance pooled across groups \( k \) and then across coordinates of \( X, j \), in LDA. Combining these two ideas, one might even invent a two-parameter set of fitted covariance matrices

\[
\hat{\Sigma}_k (\alpha, \gamma) = \alpha \hat{\Sigma}_k + (1 - \alpha) \left( \gamma \hat{\Sigma}_{pooled} + (1 - \gamma) \sigma^2 I \right)
\]

for use in QDA. Employing these in LDA or QDA provides the flexibility of choosing a complexity parameter or parameters and potentially improving prediction performance.

Returning specifically to LDA, let

\[
\bar{\mu} = \frac{1}{K} \sum_{k=1}^{K} \mu_k
\]
and note that one is free to replace $X$ and all $k$ means $\mu_k$ with respectively

$$X^* = \Sigma^{-1/2}(X - \overline{X}) \quad \text{and} \quad \mu_k^* = \Sigma^{-1/2}(\mu_k - \overline{X})$$

This produces

$$\ln \left( \frac{P[Y^* = k|X^* = x^*]}{P[Y^* = l|X^* = x^*]} \right) = \ln \left( \frac{\pi_k}{\pi_l} \right) - \frac{1}{2} \|x^* - \mu^*_k\|^2 + \frac{1}{2} \|x^* - \mu^*_l\|^2$$

and the optimal predictor/decision rule can be described as

$$\hat{Y}^*(X) = \text{arg max}_k \left[ \ln(\pi_k) - \frac{1}{2} \|X^* - \mu^*_k\|^2 \right]$$

That is, in terms of $X^*$, optimal decisions are based on ordinary Euclidian distances to the transformed means $\mu^*_k$.

The $\mu^*_k$ typically span a subspace of $\mathbb{R}^p$ of dimension $\min(p, K - 1)$. For $M_{p \times K} = (\mu^*_1, \mu^*_2, \ldots, \mu^*_K)$ let $P_M$ be the $p \times p$ projection matrix projecting onto the column space of $M$ in $\mathbb{R}^p (C(M))$. Then

$$\|x^* - \mu^*_k\|^2 = \|P_MM(x^* - \mu^*_k)\|^2$$

$$= \|P_Mx^* - \mu^*_k\|^2 + \|(I - P_M)x^*\|^2$$

the last equality coming because $(P_Mx^* - \mu^*_k) \in C(M)$ and $(I - P_M)x^* \in C(M)^\perp$. Since $\|(I - P_M)x^*\|^2$ doesn’t depend upon $k$, the optimal predictor/decision rule can be described as

$$\hat{Y}^*(X) = \text{arg max}_k \left[ \ln(\pi_k) - \frac{1}{2} \|P_MX^* - \mu^*_k\|^2 \right]$$

and optimal decision rules can be described in terms of the projection of $X^*$ onto $C(M)$ and its distances to the $\mu^*_k$.

Now,

$$\frac{1}{K}MM'$$

is the (typically rank $K - 1$) sample covariance matrix of the $\mu^*_k$ and has an eigen decomposition as

$$\frac{1}{K}MM' = VDV'$$

for

$$D = \text{diag}(d_1, d_2, \ldots, d_p)$$

where

$$d_1 \geq d_2 \geq \cdots \geq d_p$$
are the eigenvalues and the columns of $V$ are orthonormal eigenvectors corresponding in order to the successively smaller eigenvalues of $\frac{1}{K}MM'$. These $v_k$ with $d_k > 0$ specify linear combinations of the coordinates of the $\mu^*_k$, $\langle v, \mu^*_k \rangle$, with the largest possible sample variances subject to the constraints that $\|v\| = 1$ and $\langle v_l, v_k \rangle = 0$ for all $l < k$. These $v_l$ are $\perp$ vectors in directions of most important unaccounted-for spread of the $\mu^*_k$. This suggests the possibility of "reduced rank LDA."

That is, define

$$V_l = (v_1, v_2, \ldots, v_l)$$

let

$$P_l = V_lV_l'$$

be the matrix projecting onto $C(V_l)$ in $\mathbb{R}^p$. A possible "reduced rank" version of the LDA classification rule is

$$\hat{Y}_l^* (X) = \arg \max_k \left[ \ln (\pi_k) - \frac{1}{2} \| P_lX - P_l\mu^*_k \|^2 \right]$$

and $l$ becomes a complexity parameter that one might optimize to tune or regularize the method.

Note also that for $w \in \mathbb{R}^p$

$$P_lw = \sum_{k=1}^l \langle v_k, w \rangle v_k$$

For purposes of graphical representation of what is going on in these computations, one might replace the $p$ coordinates of $X$ and the means $\mu_k$ with the $\min (p, K - 1)$ coordinates of

$$\left( \langle v_1, X^* \rangle, \langle v_2, X^* \rangle, \ldots, \langle v_{\min(p, K-1)}, X^* \rangle \right)'$$ (19)

and of the

$$\left( \langle v_1, \mu^*_k \rangle, \langle v_2, \mu^*_k \rangle, \ldots, \langle v_{\min(p, K-1)}, \mu^*_k \rangle \right)'$$ (20)

It seems to be essentially ordered pairs of these coordinates that are plotted in HTF (in their Figures 4.8 and 4.11). In this regard, we need to point out that since any eigenvector $v_k$ could be replaced by $-v_k$ without any fundamental effect in the above development, the vector (19) and all of the vectors (20) could be altered by multiplication of any particular set of coordinates by $-1$. (Whether a particular algorithm for finding eigenvectors produces $v_k$ or $-v_k$ is not fundamental, and there seems to be no standard convention in this regard. It appears that the pictures in HTF might have been made using the R function lda and its choice of signs for eigenvectors.)

### 3.2 Logistic Regression

A generalization of the MVN conditional distribution result (18) is an assumption that for all $k < K$

$$\ln \left( \frac{P | Y^* = k | X = x}{P | Y^* = K | X = x} \right) = \beta_{k0} + \beta_k'x$$ (21)
Here there are \( K - 1 \) constants \( \beta_{k0} \) and \( K - 1 \) \( p \)-vectors \( \beta_k \) to be specified, not necessarily tied to class mean vectors or a common within-class covariance matrix for \( X \). In fact, the set of relationships (21) do not fully specify a joint distribution for \((X, Y)\). Rather, they only specify the nature of the conditional distributions of \( Y|X = x \). (In this regard, the situation is exactly analogous to that in ordinary simple linear regression. A bivariate normal distribution for \((X, Y)\) gets one normal conditional distributions for \( Y \) with a constant variance and mean linear in \( X \). But one may make those assumptions conditionally on \( X \), without assuming anything about the marginal distribution of \( X \), that in the bivariate normal model is univariate normal.) Using \( \theta \) as shorthand for a vector containing all the constants \( \beta_{k0} \) and the vectors \( \beta_k \), the linear log odds assumption (21) produces the forms

\[
P[Y^* = k|X = x] = p_k(x, \theta) = \frac{\exp(\beta_{k0} + \beta_k'x)}{1 + \sum_{k=1}^{K-1} \exp(\beta_{k0} + \beta_k'x)}
\]

for \( k < K \), and

\[
P[Y^* = K|X = x] = p_K(x, \theta) = \frac{1}{1 + \sum_{k=1}^{K-1} \exp(\beta_{k0} + \beta_k'x)}
\]

and the corresponding optimal predictor/classification rule is

\[
\hat{Y}^*(X) = \arg \max_k P[Y^* = k|X = x]
\]

Not only does assumption (21) generalize the "mixture of MVNs" assumption of LDA, but the standard methods of fitting the corresponding parameters based on training data are necessarily fundamentally different. That is (using maximum likelihood) in LDA, the \( K \) probabilities \( \pi_k \), the \( K \) means \( \mu_k \), and the covariance matrix \( \Sigma \) are chosen to maximize the likelihood

\[
\prod_{i=1}^{N} \pi_{Y_i^*} f(X_i|\mu_{Y_i^*}, \Sigma)
\]

This is a mixture model and the complete likelihood is involved, i.e. a joint density for the \( N \) pairs \((X_i, Y_i^*)\).

Logistic regression methodology maximizes

\[
\prod_{i=1}^{N} p_{Y_i^*}(X_i, \theta)
\]

over choices of \( \theta \). This is not a full likelihood, but rather one conditional on the \( X_i \) observed.

### 3.3 Separating Hyperplanes

In the \( K = 2 \) group case, if there is a \( \beta \in \mathbb{R}^p \) and real number \( \beta_0 \) such that in the training data

\[
Y^* = 2 \text{ exactly when } \beta'X + \beta_0 > 0
\]
a "separating hyperplane"

\[
\{ \mathbf{x} \in \mathbb{R}^p | \mathbf{\beta}' \mathbf{x} + \beta_0 = 0 \}
\]

can be found via logistic regression. The (conditional) likelihood will not have a maximum, but if one follows a search path far enough toward limiting value of 0 for the loglikelihood or 1 for the likelihood, satisfactory \( \beta \in \mathbb{R}^p \) and \( \beta_0 \) from an iteration of the search algorithm will produce separation.

A famous older algorithm for finding a separating hyperplane is the so-called "perceptron" algorithm. It can be defined as follows. Temporarily treat \( Y^* \) taking values in \( G = \{1, 2\} \) as real-valued, and define

\[
Y^{**} = 2Y^* - 3
\]

(so that \( Y^{**} \) takes real values ±1). From some starting points \( \beta^0 \) and \( \beta_0^0 \) cycle through the training data cases in order (repeatedly as needed). At any iteration \( l \), take

\[
\begin{align*}
\{ \beta^l = \beta^{l-1} \text{ and } \beta_0^l = \beta_{0}^{l-1} \} & \quad \text{if } \left\{ \begin{array}{l}
Y_i^* = 2 \text{ and } \beta' \mathbf{X}_i + \beta_0 > 0, \text{ or } \\
Y_i^* = 1 \text{ and } \beta' \mathbf{X}_i + \beta_0 \leq 0
\end{array} \right. \\
\{ \beta^l = \beta^{l-1} + Y_i^* \mathbf{X}_i \text{ and } \beta_0^l = \beta_{0}^{l-1} + Y_i^* \} & \quad \text{otherwise}
\end{align*}
\]

This will eventually identify a separating hyperplane when a series of \( N \) iterations fails to change the values of \( \beta \) and \( \beta_0 \).

If there is a separating hyperplane, it will not be unique. One can attempt to define and search for "optimal" such hyperplanes that, e.g., maximize distance from the plane to the closest training vector. See HTF Section 4.5.2 in this regard.

### 4 Basis Expansions and Regularization (Ch 5 of HTF)

A fundamental way of moving beyond prediction rules that are functions of a linear form in \( \mathbf{X} \), i.e. depend upon \( \mathbf{X} \) through \( \mathbf{\beta}' \mathbf{X} = \sum_{j=1}^{p} \beta_j X_j \) is to consider some set of (basis) functions \( h_m \) and predictors of the form or depending upon the form

\[
f(\mathbf{X}) = \sum_{m=1}^{p} \beta_m h_m(\mathbf{X}) \quad (22)
\]

We next consider some flexible methods employing this idea. Until further notice, we restrict attention to a one-dimensional input variable \( X \). Notice that fitting of form (22) can be done using OLS (or any of the methods of HTF Chapter 3) based on the \( N \times p \) matrix of inputs

\[
\mathbf{X} = (h_j(x_i))
\]

\((i \text{ indexing rows and } j \text{ indexing columns})\).
4.1 Piecewise Polynomials and Regression Splines

Consider \( K \) "knots" \( \xi_1 < \xi_2 < \cdots < \xi_K \)

and forms for \( f(x) \) that are

1. polynomials of order \( M \) (or less) on all intervals \((\xi_{j-1}, \xi_j)\), and (potentially, at least)

2. have derivatives of some specified order at the knots, and (potentially, at least)

3. are linear outside \((\xi_1, \xi_K)\).

If we let \( I_1(x) = I[x < \xi_1] \), for \( j = 2, \ldots, K \) let \( I_j(x) = I[\xi_{j-1} \leq x < \xi_j] \), and define \( I_{K+1}(x) = I[\xi_K \leq x] \), one can have 1. in the list above using basis functions

\[
I_1(x), I_2(x), \ldots, I_{K+1}(x) \\
xI_1(x), xI_2(x), \ldots, xI_{K+1}(x) \\
x^2I_1(x), x^2I_2(x), \ldots, x^2I_{K+1}(x) \\
\vdots \\
x^MI_1(x), x^MI_2(x), \ldots, x^MI_{K+1}(x)
\]

Further, one can enforce continuity and differentiability (at the knots) conditions on a form \( f(x) = \sum_{m=1}^{(M+1)(K+1)} \beta_m h_m(x) \) by enforcing some linear relations between appropriate ones of the \( \beta_m \). While this is conceptually simple, it is messy. It is much cleaner to simply begin with a set of basis functions that are tailored to have the desired continuity/differentiability properties.

A set of \( M + 1 + K \) basis functions for piecewise polynomials of degree \( M \) with derivatives of order \( M - 1 \) at all knots is easily seen to be

\[
1, x, x^2, \ldots, x^M, (x - \xi_1)^M_+, (x - \xi_2)^M_+, \ldots, (x - \xi_K)^M_+
\]

(since the value of and first \( j \) derivatives of \((x - \xi_j)^M_+\) at \( \xi_j \) are all 0). The choice of \( M = 3 \) is fairly standard.

Since extrapolation with polynomials typically gets worse with order, it is common to impose a restriction that outside \((\xi_1, \xi_K)\) a form \( f(x) \) be linear. For the case of \( M = 3 \) this can be accomplished by beginning with basis functions \( 1, x, (x - \xi_1)^3_+, (x - \xi_2)^3_+, \ldots, (x - \xi_K)^3_+ \) and imposing restrictions necessary to force 2nd and 3rd derivatives to the right of \( \xi_K \) to be 0. Notice that (considering \( x > \xi_K \))

\[
\frac{d^2}{dx^2} \left( \alpha_0 + \alpha_1 + \sum_{j=1}^{K} \beta_j (x - \xi_j)^3_+ \right) = 6 \sum_{j=1}^{K} \beta_j (x - \xi_j)
\]

(23)
and
\[
\frac{d^3}{dx^3} \left( \alpha_0 + \alpha_1 + \sum_{j=1}^{K} \beta_j (x - \xi_j)_+^3 \right) = 6 \sum_{j=1}^{K} \beta_j
\]  
(24)

So, linearity for large \( x \) requires (from (24)) that \( \sum_{j=1}^{K} \beta_j = 0 \). Further, substituting this into (23) means that linearity also requires that \( \sum_{j=1}^{K} \beta_j \xi_j = 0 \).

Using the first of these to conclude that \( \beta_K = -\sum_{j=1}^{K-1} \beta_j \) and substituting into the second yields
\[
\beta_{K-1} = -\sum_{j=1}^{K-2} \beta_j \left( \frac{\xi_K - \xi_j}{\xi_K - \xi_{K-1}} \right)
\]

and then
\[
\beta_K = \sum_{j=1}^{K-2} \beta_j \left( \frac{\xi_K - \xi_j}{\xi_K - \xi_{K-1}} \right) - \sum_{j=1}^{K-2} \beta_j
\]

These then suggest the set of basis functions consisting of \( 1, x \) and for \( j = 1, 2, \ldots, K - 2 \)
\[
(x - \xi_j)_+^3 - \left( \frac{\xi_K - \xi_j}{\xi_K - \xi_{K-1}} \right) (x - \xi_{K-1})_+^3 - (x - \xi_K)_+^3 + \left( \frac{\xi_K - \xi_j}{\xi_K - \xi_{K-1}} \right) (x - \xi_K)_+^3
\]

(These are essentially the basis functions that HTF call their \( N_j \).) Their use produces so-called "natural" (linear outside \((\xi_1, \xi_K)\)) cubic regression splines.

There are other (harder to motivate, but in the end more pleasing and computationally more attractive) sets of basis functions for natural polynomial splines. See the B-spline material at the end of HTF Chapter 5.

### 4.2 Smoothing Splines

A way of avoiding the direct selection of knots is to instead, for a smoothing parameter \( \lambda > 0 \), consider the problem of finding

\[
f_\lambda = \arg \min_{\text{functions } h \text{ with 2 derivatives}} \left( \sum_{i=1}^{N} (y_i - h(x_i))^2 + \lambda \int |h''(x)|^2 \, dx \right)
\]

Amazingly enough, this optimization problem has a solution that can be fairly simply described. \( f_\lambda \) is a natural cubic spline with knots at the distinct values \( x_i \) in the training set. That is, for a set of basis functions for such splines

\[
h_1, h_2, \ldots, h_N
\]

(here we’re tacitly assuming that the \( N \) values of the input variable in the training set are all different)

\[
f_\lambda(x) = \sum_{j=1}^{N} \tilde{\beta}_j h_j(x)
\]
where the $\hat{\beta}_{\lambda j}$ are yet to be identified.

So consider the function

$$g(x) = \sum_{j=1}^{N} \theta_j h_j(x) \quad (25)$$

this has second derivative

$$g''(x) = \sum_{j=1}^{N} \theta_j h_j''(x)$$

and so

$$\left(g''(x)\right)^2 = \sum_{j=1}^{N} \sum_{l=1}^{N} \theta_j \theta_l h_j''(x) h_l''(x)$$

$$= \theta'\Omega\theta$$

for $\theta' = (\theta_1, \theta_2, \ldots, \theta_N)$ and

$$\Omega_{N\times N} = \left(\int h_j''(t) h_l''(t) \, dt\right)$$

In fact, with the notation

$$H_{N\times N} = (h_j(x))$$

($i$ indexing rows and $j$ indexing columns) the criterion to be optimized in order to find $f_\lambda$ can be written for functions of the form (25) as

$$(Y - H\theta)'(Y - H\theta) + \lambda \theta' \Omega \theta$$

and some vector calculus shows that the optimizing $\theta$ is

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

which can be thought of as some kind of vector of generalized ridge regression coefficients.

Corresponding to (26) is a vector of smoothed output values

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

and the matrix

$$S_\lambda = H (H'H + \lambda \Omega)^{-1} H'$$

is called a smoother matrix. Contrast this to a situation where some fairly small number, $p$, of basis functions are employed in a regression context. That is, for basis function $b_1, b_2, \ldots, b_p$ suppose

$$B_{N\times p} = (b_j(x_i))$$
Then OLS produces the vector of fitted values

$$\hat{Y} = B (B' B)^{-1} B' Y$$

and the projection matrix onto the column space of $B$, $C(B)$, is $P_B = B (B' B)^{-1} B'$. $S_\lambda$ and $P_B$ are both $N \times N$ symmetric non-negative definite matrices. While

$$P_B P_B = P_B$$
i.e. $P_B$ is idempotent,

$$S_\lambda S_\lambda \preceq S_\lambda$$

meaning that $S_\lambda - S_\lambda S_\lambda$ is non-negative definite. $P_B$ is of rank $p = \text{tr}(P_B)$, while $S_\lambda$ is of rank $N$. In a manner similar to what is done in ridge regression we might define an "effective degrees of freedom" for $S_\lambda$ (or for smoothing) as

$$\text{df}(\lambda) = \text{tr}(S_\lambda)$$

We proceed to develop motivation and a formula for this quantity.

Notice that it is possible to write

$$S_\lambda^{-1} = I + \lambda K$$

for

$$K = (H')^{-1} \Omega H^{-1}$$

so that

$$S_\lambda = (I + \lambda K)^{-1}$$

Some vector calculus shows that $\hat{Y}_\lambda = S_\lambda Y$ is a solution to the minimization problem

$$\min_{v \in \mathbb{R}^N} \left( (Y - v)' (Y - v) + \lambda v' K v \right)$$

so that this matrix $K$ can be thought of as defining a "penalty" in fitting a smoothed version of $Y$.

Then, since $S_\lambda$ is symmetric non-negative definite, it has an eigen decomposition as

$$S_\lambda = U D U' = \sum_{j=1}^{N} d_j u_j u_j'$$

where columns of $U$ (the eigenvectors $u_j$) comprise an orthonormal basis for $\mathbb{R}^N$ and

$$D = \text{diag}(d_1, d_2, \ldots, d_N)$$

for eigenvalues of $S_\lambda$

$$d_1 \geq d_2 \geq \cdots \geq d_N > 0$$

It turns out to be guaranteed that $d_1 = d_2 = 1$. Consider how the eigenvalues and eigenvectors of $S_\lambda$ are related to those for $K$. 

31
An eigenvalue for $K$, say $\eta$, solves

$$\det (K - \eta I) = 0$$

Now

$$\det (K - \eta I) = \det \left( \frac{1}{\lambda} [(I + \lambda K) - (1 + \lambda \eta) I] \right)$$

So $1 + \lambda \eta$ must be an eigenvalue of $I + \lambda K$ and $1/ (1 + \lambda \eta)$ must be an eigenvalue of $S_\lambda = (I + \lambda K)^{-1}$. So for some $j$ we must have

$$d_j = \frac{1}{1 + \lambda \eta}$$

and observing that $1/ (1 + \lambda \eta)$ is decreasing in $\eta$, we may conclude that

$$d_j = \frac{1}{1 + \lambda \eta_{N-j+1}}$$

for

$$\eta_1 \geq \eta_2 \geq \cdots \geq \eta_{N-2} \geq \eta_{N-1} = \eta_N = 0$$

the eigenvalues of $K$ (that themselves do not depend upon $\lambda$). So, for example, in light of (27), (30), and (31), the smoothing effective degrees of freedom are

$$\text{df}(\lambda) = \text{tr}(S_\lambda) = \text{tr} \left( \sum_{j=1}^{N} d_j u_j u_j' \right) = \text{tr} \left( \sum_{j=1}^{N} d_j u_j u_j' \right) = \sum_{j=1}^{N} d_j = \sum_{j=1}^{N} \frac{1}{1 + \lambda \eta_j}$$

which is clearly decreasing in $\lambda$ (with minimum value 2 in light of the fact that $S_\lambda$ has two eigenvalues that are 1).

Further, consider $u_j$, the eigenvector of $S_\lambda$ corresponding to eigenvalue $d_j$. $S_\lambda u_j = d_j u_j$ so that

$$u_j = S_\lambda^{-1} d_j u_j = (I + \lambda K) d_j u_j$$

so that

$$u_j = d_j u_j + d_j \lambda K u_j$$

and thus

$$K u_j = \left( \frac{1 - d_j}{\lambda d_j} \right) u_j = \eta_{N-j+1} u_j$$

That is, $u_j$ is an eigenvector of $K$ corresponding to the $(N - j + 1)$st largest eigenvalue. That is, for all $\lambda$ the eigenvectors of $S_\lambda$ are eigenvectors of $K$ and thus do not depend upon $\lambda$.

Then, for any $\lambda$

$$\tilde{Y}_\lambda = S_\lambda Y = \left( \sum_{j=1}^{N} d_j u_j u_j' \right) Y = \sum_{j=1}^{N} d_j \langle u_j, Y \rangle u_j = \sum_{j=1}^{N} \frac{\langle u_j, Y \rangle}{1 + \lambda \eta_{N-j+1}} u_j$$

(32)
and we see that $\hat{Y}_\lambda$ is a shrunken version of $Y$. The larger is $\lambda$, the more severe the shrinking overall. Further, the larger is $j$, the smaller is $d_j$ and the more severe is the shrinking in the $u_j$ direction. In the context of cubic smoothing splines, large $j$ correspond to "wiggly" (as a functions of coordinate $i$ or value of the input $x_i$) $u_j$, and the prescription (32) calls for suppression of "wiggly" components of $Y$.

Notice that large $j$ correspond to early/large eigenvalues of the penalty matrix $K$ in (29). Letting $u_j^* = u_{N-j+1}$ so that

$$U^* = (u_N, u_{N-1}, \ldots, u_1)$$

$$= (u_1^*, u_2^*, \ldots, u_N^*)$$

the eigen decomposition of $K$ is

$$K = U^* \text{diag}(\eta_1, \eta_2, \ldots, \eta_N) U^{*'}$$

and the criterion (29) can be written as

$$\min_{v \in \mathbb{R}^N} (Y - v)'(Y - v) + \lambda v' U^* \text{diag}(\eta_1, \eta_2, \ldots, \eta_N) U^{*'} v$$

or equivalently as

$$\min_{v \in \mathbb{R}^N} \left( (Y - v)'(Y - v) + \lambda \sum_{j=1}^{N} \eta_j \langle u_j^*, v \rangle^2 \right) \quad (33)$$

and we see that eigenvalues of $K$ function as penalty coefficients applied to the $N$ orthogonal components of $v = \sum_{j=1}^{N} \langle u_j^*, v \rangle u_j^*$ in the choice of optimizing $v$. From this point of view, the $u_j$ (or $u_j^*$) provide the natural alternative (to the columns of $H$) basis (for $\mathbb{R}^N$) for representing or approximating $Y$, and the last equality in (32) provides an explicit form for the optimizing smoothed vector $\hat{Y}_\lambda$.

In this development, $K$ has had a specific meaning derived from the $H$ and $\Omega$ matrices connected specifically with smoothing splines. But in the end, perhaps the most interesting possibility brought up by the whole development is that of forgetting the origins (from $K$) of the $\eta_j$ and $u_j$ and beginning with any interesting/intuitively appealing orthonormal basis $\{u_j\}$ and set of non-negative penalties $\{\eta_j\}$ for use in (33). Working backwards through (32) and (31) one is then led to the corresponding smoothed vector $\hat{Y}_\lambda$ and smoothing matrix $S_\lambda$.

It is also worth remarking that since $\hat{Y}_\lambda = S_\lambda Y$ the rows of $S_\lambda$ provide weights to be applied to the elements of $Y$ in order to produce predictions/smoothed values corresponding to $Y$. These can for each $i$ be thought of as defining a corresponding "equivalent kernel" (for an appropriate "kernel-weighted average" of the training output values). (See Figure 5.8 of HTF2 in this regard.)
4.3 Generalizations to p-Dimensional Inputs

4.3.1 Multi-Dimensional Regression Splines (Tensor Bases)

If $p = 2$ and the vector of inputs, $X$, takes values in $\mathbb{R}^2$, one might proceed as follows. If $\{h_{11}, h_{12}, \ldots, h_{1M_1}\}$ is a set of spline basis functions based on $X_1$ and $\{h_{21}, h_{22}, \ldots, h_{2M_2}\}$ is a set of spline basis functions based on $X_2$ on might consider the set of $M_1 \cdot M_2$ basis functions based on $X$ defined by

$$g_{jk}(x) = h_{1j}(x_1) h_{2k}(x_2)$$

and corresponding regression splines

$$f(x) = \sum_{j,k} \beta_{jk} g_{jk}(x)$$

The biggest problem with this potential method is the explosion in the size of a tensor product basis as $p$ increases. For example, using $K$ knots for cubic regression splines in each of $p$ dimensions produces $(4 + K)^p$ basis functions for the $p$-dimensional problem. Some kind of forward selection algorithm or shrinking of coefficients will be needed to produce any kind of workable fits with such large numbers of basis functions.

4.3.2 Multi-Dimensional Smoothing Splines

If $p = 2$ and the vector of inputs, $X$, takes values in $\mathbb{R}^2$, one might propose to seek

$$f_\lambda = \arg\min_{\text{functions } h \text{ with 2 derivatives}} \left( \sum_{i=1}^{N} (y_i - h(x_i))^2 + \lambda J[h] \right)$$

for

$$J[h] \equiv \int_{y^2} \left( \frac{\partial^2 h}{\partial x_1^2} \right)^2 + 2 \left( \frac{\partial^2 h}{\partial x_1 \partial x_2} \right)^2 + \left( \frac{\partial^2 h}{\partial x_2^2} \right)^2 \, dx_1 dx_2$$

An optimizing $f_\lambda : \mathbb{R}^2 \rightarrow \mathbb{R}$ can be identified and is called a "thin plate spline." As $\lambda \rightarrow 0$, $f_\lambda$ becomes an interpolator, as $\lambda \rightarrow \infty$ it defines the OLS plane through the data in 3-space. In general, it can be shown to be of the form

$$f_\lambda(x) = \beta_{0\lambda} + \beta'_\lambda x + \sum_{i=1}^{N} \alpha_i g_i(x) \quad (34)$$

where $g_i(x) = \eta(||x - x_i||)$ for $\eta(z) = z^2 \ln z^2$. The $g_i(x)$ are "radial basis functions" (radially symmetric basis functions) and fitting is accomplished much as for the $p = 1$ case. The form (34) is plugged into the optimization criterion and a discrete penalized least squares problem emerges (after taking account of some linear constraints that are required to keep $J[f_\lambda] < \infty$. HTF seem to indicate that in order to keep computations from exploding with $N$, it usually suffices to replace the $N$ functions $g_i(x)$ in (34) with $K \ll N$ functions $g_i^*(x) = \ldots$
\eta(||x - x_i^*||) for K potential input vectors \(x_i^*\) placed on a rectangular grid covering the convex hull of the \(N\) training data input vectors \(x_i\).

For large \(p\), one might simply declare that attention is going to be limited to predictors of some restricted form, and for \(h\) in that restricted class, seek to optimize

\[
\sum_{i=1}^{N} (y_i - h(x_i))^2 + \lambda J[h]
\]

for \(J[h]\) some appropriate penalty on \(h\) intended to regularize/restrict its wiggling. For example, one might assume that a form

\[
g(x) = \sum_{j=1}^{p} g_j(x_j)
\]

and set

\[
J[g] = \sum_{j=1}^{p} \int (g_j''(x))^2 dx
\]

and be led to additive splines.

Or, one might assume that

\[
g(x) = \sum_{j=1}^{p} g_j(x_j) + \sum_{j,k} g_{jk}(x_j, x_k)
\]

and invent an appropriate penalty function. It seems like a sum of 1-d smoothing spline penalties on the \(g_j\) and 2-d thin plate spline penalties on the \(g_{jk}\) is the most obvious starting point. Details of fitting are a bit murky (though I am sure that they can be found in book on generalized additive models). Presumably one cycles through the summands in (35) iteratively fitting functions to sets of residuals defined by the original \(y_i\) minus the sums of all other current versions of the components until some convergence criterion is satisfied. (35) is a kind of "main effects plus 2-factor interactions" decomposition, but it is (at least in theory) possible to also consider higher order terms in this kind of expansion.

4.4 Reproducing Kernel Hilbert Spaces and Fitting Functions in \(\mathbb{R}^p\)

A general framework that encompasses many regularized fitting methods is that of reproducing kernel Hilbert spaces. This begins with \(C\) a compact subset of \(\mathbb{R}^p\) and a symmetric kernel function

\[
K: C \times C \rightarrow \mathbb{R}
\]

Ultimately, we will consider as predictors for \(x \in C\) linear combinations of sections of the kernel function, \(\sum_{i=1}^{N} b_i K(x, x_i)\) (where the \(x_i\) are the input vectors
in the training set). But to get there in a rational way, and to incorporate use of a complexity penalty into the fitting, we will restrict attention to those kernels that have nice properties. In particular, we require that \( K \) be continuous and non-negative definite. We mean by this latter that for any set of \( M \) elements of \( \mathbb{R}^p \), say \( \{x_j\} \), the \( M \times M \) matrix

\[
(K(x_i, x_j))
\]

is non-negative definite.

Then according to what is known as Mercer’s Theorem (look it up in Wikipedia) \( K \) may then be written in the form

\[
K(z, x) = \sum_{i=1}^{\infty} \gamma_i \phi_i(z) \phi_i(x)
\]

(36)

for linearly independent (in \( L^2(C) \)) functions \( \{\phi_i\} \), and constants \( \gamma_i \geq 0 \), where the \( \phi_i \) comprise an orthonormal basis for \( L^2(C) \) (so any function in \( L^2(C) \) has an expansion in terms of the \( \phi_i \)), and the ones corresponding to positive \( \gamma_i \) may be taken to be continuous on \( C \). Further, the \( \phi_i \) are "eigenfunctions" of the kernel \( K \) corresponding to the "eigenvalues \( \gamma_i \)" in the sense that in \( L^2(C) \)

\[
\int \phi_i(z) K(z, \cdot) dz = \gamma_i \phi_i(\cdot)
\]

Our present interest is in a function space \( \mathcal{H}_K \) (that is a subset of \( L^2(C) \) with a different inner product and norm) with members of the form

\[
f(x) = \sum_{i=1}^{\infty} c_i \phi_i(x) \quad \text{for } c_i \text{ with } \sum_{i=1}^{\infty} \frac{c_i^2}{\gamma_i} < \infty
\]

(37)

(called the "primal form" of functions in the space). (Notice that all elements of \( L^2(C) \) are of the form \( f(x) = \sum_{i=1}^{\infty} c_i \phi_i(x) \) with \( \sum_{i=1}^{\infty} c_i^2 < \infty \).) More naturally, our interest centers on

\[
f(x) = \sum_{i=1}^{\infty} b_i K(x, z_i) \quad \text{for some set of } z_i
\]

(38)

supposing that the series converges appropriately (called the "dual form" of functions in the space). The former is most useful for producing simple proofs, while the second is most natural for application, since how to obtain the \( \phi_i \) and corresponding \( \gamma_i \) for a given \( K \) is not so obvious. Notice that

\[
K(z, x) = \sum_{i=1}^{\infty} \gamma_i \phi_i(z) \phi_i(x)
\]

\[
= \sum_{i=1}^{\infty} (\gamma_i \phi_i(x)) \phi_i(z)
\]
and letting $\gamma_i \phi_i (x) = c_i (x)$, since $\sum_{i=1}^{\infty} c_i^2 (x) / \gamma_i = \sum_{i=1}^{\infty} \gamma_i \phi_i^2 (x) = K (x, x) < \infty$, the function $K (\cdot, x)$ is of the form (37), so that we can expect functions of the form (38) with absolutely convergent $\sum_{i=1}^{\infty} b_i$ to be of form (37).

In the space of functions (37), we define an inner product (for our Hilbert space)

$$\left\langle \sum_{i=1}^{\infty} c_i \phi_i, \sum_{i=1}^{\infty} d_i \phi_i \right\rangle_{\mathcal{H}_K} = \sum_{i=1}^{\infty} \frac{c_i d_i}{\gamma_i}$$

so that

$$\left\| \sum_{i=1}^{\infty} c_i \phi_i \right\|^2_{\mathcal{H}_K} = \sum_{i=1}^{\infty} \frac{c_i^2}{\gamma_i}$$

Note then that for $f = \sum_{i=1}^{\infty} c_i \phi_i$ belonging to our Hilbert space $\mathcal{H}_K$,

$$\langle f, K (\cdot, x) \rangle_{\mathcal{H}_K} = \sum_{i=1}^{\infty} \frac{c_i \gamma_i \phi_i (x)}{\gamma_i} = \sum_{i=1}^{\infty} c_i \phi_i (x) = f (x)$$

and so $K (\cdot, x)$ is called the representer of evaluation at $x$. Further,

$$\langle K (\cdot, z), K (\cdot, x) \rangle_{\mathcal{H}_K} = K (z, x)$$

which is the reproducing property of the RKHS. Notice that this reproducing property implies that for $f (x) = \sum_{i=1}^{M} b_i K (x, z_i)$ for some set of $z_i$,

$$\|f\|^2_{\mathcal{H}_K} = \langle f, f \rangle_{\mathcal{H}_K} = \sum_{i=1}^{M} \sum_{j=1}^{M} b_i b_j K (z_i, z_j)$$

That is, a function that is a linear combination of some set of slices of the kernel function has squared norm in $\mathcal{H}_K$ that is the variance of that linear combination of coordinates of a random vector with covariance matrix obtained by consulting the kernel at pairs of points $z_i$ that specify which slices are used to make up $f$.

For applying this material to the fitting of training data, for $\lambda > 0$ and a loss function $L (y, \hat{y}) \geq 0$ define an optimization criterion

$$\min_{f \in \mathcal{H}_K} \left( \sum_{i=1}^{N} L(y_i, f (x_i)) + \lambda \|f\|^2_{\mathcal{H}_K} \right)$$

(39)

As it turns out, an optimizer of this criterion must, for the training vectors $\{x_i\}$, be of the form

$$f (x) = \sum_{i=1}^{N} b_i K (x, x_i)$$

(40)

and the corresponding $\|f\|^2_{\mathcal{H}_K}$ is then

$$\langle f, f \rangle_{\mathcal{H}_K} = \sum_{i=1}^{N} \sum_{j=1}^{N} b_i b_j K (x_i, x_j)$$

37
The criterion (39) is thus

\[ \minimize_{b \in \mathbb{R}^N} \left( \sum_{i=1}^{N} L \left( y_i, \sum_{j=1}^{N} b_j K(x_i, x_j) \right) + \lambda b' (K(x_i, x_j)) b \right) \]  

(41)

Letting

\[ K = (K(x_i, x_j)) \quad \text{and} \quad P = K^{-} \quad \text{(a symmetric generalized inverse of} \ K) \]

and defining

\[ L_N(Y, Kb) = \sum_{i=1}^{N} L \left( y_i, \sum_{j=1}^{N} b_j K(x_i, x_j) \right) \]

the optimization criterion (41) is

\[ \minimize_{b \in \mathbb{R}^N} (L_N(Y, Kb) + \lambda b' K b) \]

i.e.

\[ \minimize_{b \in \mathbb{R}^N} (L_N(Y, Kb) + \lambda b' K' P K b) \]

i.e.

\[ \minimize_{v \in C(K)} (L_N(Y, v) + \lambda v' P v) \]  

(42)

That is, the function space optimization problem (39) reduces to the \(N\)-dimensional optimization problem (42). A \( v_\lambda \in C(K) \) (the column space of \( K \)) minimizing \( L_N(Y, v) + \lambda v' P v \) corresponds to \( b_\lambda \) minimizing \( L_N(Y, Kb) + \lambda b' K b \) via

\[ K b_\lambda = v_\lambda \]  

(43)

For the particular special case of squared error loss, \( L(y, \hat{y}) = (y - \hat{y})^2 \), this development has a very explicit punch line. That is,

\[ L_N(Y, Kb) + \lambda b' K b = (Y - Kb)' (Y - Kb) + \lambda b' K b \]

Some vector calculus shows that this is minimized over choices of \( b \) by

\[ b_\lambda = (K + \lambda I)^{-1} Y \]  

(44)

and corresponding fitted values are

\[ \hat{Y}_\lambda = v_\lambda = K (K + \lambda I)^{-1} Y \]

Then using (44) under squared error loss, the solution to (39) is from (40)

\[ f_\lambda(x) = \sum_{i=1}^{N} b_\lambda L(x, x_i) \]  

(45)
To better understand the nature of (42), consider the eigen decomposition of $K$ in a case where it is non-singular as

$$K = U \text{diag} (\eta_1, \eta_2, \ldots, \eta_N) U'$$

for eigenvalues $\eta_1 \geq \eta_2 \geq \cdots \geq \eta_N > 0$, where the eigenvector columns of $U$ comprise an orthonormal basis for $\mathbb{R}^N$. The penalty in (42) is

$$\lambda v' P v = \lambda v' U \text{diag} (1/\eta_1, 1/\eta_2, \ldots, 1/\eta_N) U' v$$

$$= \lambda \sum_{j=1}^{N} \frac{1}{\eta_j} \langle v, u_j \rangle^2$$

Now remembering that the $u_j$ comprise an orthonormal basis for $\mathbb{R}^N$

$$v = \sum_{j=1}^{N} \langle v, u_j \rangle u_j \quad \text{and} \quad \|v\|^2 = \langle v, v \rangle = \sum_{j=1}^{N} \langle v, u_j \rangle^2$$

so we see that in choosing a $v$ to optimize $(L_N (Y, v) + \lambda v' P v)$, we penalize highly those $v$ with large components in the directions of the late eigenvectors of $K$ (the ones corresponding to its small eigenvalues, thereby tending to suppress those features of a potential $v$).

HTF seem to say that the eigenvalues and eigenvectors of the data-dependent $N \times N$ matrix $K$ are somehow related respectively to the constants $\gamma_i$ and functions $\phi_i$ in the representation (36) of $K$ as a weighted series of products. That seems hard to understand and (even if true) certainly not obvious.

HTF also make some discussion of fitting problems that involve candidate functions of the form

$$\sum_{j=1}^{M} c_j \psi_j + f$$

where $f \in \mathcal{H}_K$ and $\psi_j$'s are somehow not penalized by the $\lambda \|v\|^2_{\mathcal{H}_K}$ penalty, since they belong to a "null space of the kernel." This is not so easy to understand. I think what is meant is that in a properly constructed (larger) RKHS space containing all $f \in \mathcal{H}_K$ and the $\psi_j$ there is an inner product that reduces to the $\mathcal{H}_K$ inner product for $f \in \mathcal{H}_K$ and under which each $\psi_j$ is perpendicular to every $\phi_i$. Then in this larger space, the projection of $\sum_{j=1}^{M} c_j \psi_j + f$ onto $\mathcal{H}_K$ is $f$ and the penalty is set in terms of the norm of this projection alone.

Some special cases of this general RKHS development can be briefly described.

### 4.4.1 Some Special Cases

One possible kernel function in $p$ dimensions is

$$K(z, x) = (1 + \langle z, x \rangle)^d$$
For fixed $x_i$, the basis functions $K(x, x_i)$ are $d$th order polynomials in the entries of $x_i$. So the fitting is in terms of such polynomials. Note that since $K(z, x)$ is relatively simple here, there seems to be a good chance of explicitly deriving a representation (36) and perhaps working out all the details of what is above in a very concrete setting.

Another standard kernel function in $p$ dimensions is

$$K(z, x) = \exp \left( -\nu \| z - x \|^2 \right)$$

and the basis functions $K(x, x_i)$ are essentially spherically symmetric normal pdfs with mean vectors $x_i$. (These are "Gaussian radial basis functions.")

A kernel function in 2 dimensions is

$$K(z, x) = \| z - x \|^2 \ln (\| z - x \|)$$

and this (for squared error loss, $L(y, \hat{y}) = (y - \hat{y})^2$) is the case of thin plate splines.

The standard development of so-called "support vector classifiers" in a 2-class context with $Y^*$ taking values $\pm 1$, uses some kernel $K(z, x)$ and predictors

$$f(x) = b_0 + \sum_{i=1}^{\infty} b_i K(x, x_i)$$

in combination with

$$L(y^*, f(x)) = [1 - y^* f(x)]_+$$

(the sign of $f(x)$ providing the classification associated with $x$).

### 4.5 Wavelet Bases

Return now for exposition purposes to the case of a one-dimensional input variable $X$, and in fact suppose that $X$ takes values in $[0, 1]$. One might consider a set of basis function for use in the form (22) that is big enough and rich enough to approximate essentially any function on $[0, 1]$. In particular, various orthonormal bases for the square integrable functions on this interval (the space of functions $L^2[0, 1]$) come to mind. One might, for example, consider using some number of functions from the Fourier basis for $L^2[0, 1]$

$$\left\{ \sqrt{2} \sin (j2\pi t) \right\}_{j=1}^{\infty} \cup \left\{ \sqrt{2} \cos (j2\pi t) \right\}_{j=1}^{\infty} \cup \{1\}$$

For example, using $M \approx N/2$ sin-cos pairs and the constant, one could consider the fitting of predictors

$$f(X) = \beta_0 + \sum_{m=1}^{M} \beta_{1m} \sin (m2\pi X) + \sum_{m=1}^{M} \beta_{2m} \cos (m2\pi X)$$

(46)
If one has training $x_i$ on an appropriate regular grid, the use of form (46) leads to orthogonality in the $N \times (2M + 1)$ matrix of values of the basis functions $X$ and simple/fast calculations.

Unless, however, one believes that $E[Y|X = x]$ is periodic, form (46) has its serious limitations. In particular, unless $M$ is very very large, a trigonometric series like (46) will typically provide a poor approximation for a function that varies at different scales on different parts of $[0,1]$, and in any case, the coefficients necessary to provide such localized variation at different scales have no obvious simple interpretations/connections to the irregular pattern of variation being described. So-called "wavelet bases" are much more useful in providing parsimonious and interpretable approximations to such functions.

The simplest wavelet basis for $L^2[0,1]$ is the Haar basis. It may be described as follows. Define

$$\varphi(x) = I[0 < x \leq 1] \quad \text{(the so-called "father" wavelet)}$$

and

$$\psi(x) = I \left[ 0 < x \leq \frac{1}{2} \right] - I \left[ \frac{1}{2} < x \leq 1 \right] \quad \text{(the so-called "mother" wavelet)}$$

Linear combinations of these functions provide all elements of $L^2[0,1]$ that are constant on $[0, \frac{1}{2}]$ and on $[\frac{1}{2}, 1]$. Write

$$\Psi_0 = \{\varphi, \psi\}$$

Next, define

$$\psi_{1,0}(x) = \sqrt{2} \left( I \left[ 0 < x \leq \frac{1}{4} \right] - I \left[ \frac{1}{4} < x \leq \frac{1}{2} \right] \right) \quad \text{and}$$

$$\psi_{1,1}(x) = \sqrt{2} \left( I \left[ \frac{1}{2} < x \leq \frac{3}{4} \right] - I \left[ \frac{3}{4} < x \leq 1 \right] \right)$$

and let

$$\Psi_1 = \{\psi_{1,0}, \psi_{1,1}\}$$

Using the set of functions $\Psi_0 \cup \Psi_1$ one can build (as linear combinations) all elements of $L^2[0,1]$ that are constant on $[0, \frac{1}{4}]$ and on $[\frac{1}{4}, \frac{1}{2}]$ and on $[\frac{1}{2}, \frac{3}{4}]$ and on $[\frac{3}{4}, 1]$.

The story then goes on as one should expect. One defines

$$\psi_{2,0}(x) = 2 \left( I \left[ 0 < x \leq \frac{1}{8} \right] - I \left[ \frac{1}{8} < x \leq \frac{1}{4} \right] \right) \quad \text{and}$$

$$\psi_{2,1}(x) = 2 \left( I \left[ \frac{1}{4} < x \leq \frac{3}{8} \right] - I \left[ \frac{3}{8} < x \leq \frac{1}{2} \right] \right) \quad \text{and}$$

$$\psi_{2,2}(x) = 2 \left( I \left[ \frac{1}{2} < x \leq \frac{5}{8} \right] - I \left[ \frac{5}{8} < x \leq \frac{3}{4} \right] \right) \quad \text{and}$$

$$\psi_{2,3}(x) = 2 \left( I \left[ \frac{3}{4} < x \leq \frac{7}{8} \right] - I \left[ \frac{7}{8} < x \leq 1 \right] \right)$$

41
and lets
\[ \Psi_2 = \{ \psi_2,0, \psi_2,1, \psi_2,2, \psi_2,3 \} \]

In general,
\[ \psi_{m,j}(x) = \sqrt{2^m} \psi \left( 2^m \left( x - \frac{j}{2^m} \right) \right) \quad \text{for} \quad j = 1, 2, \ldots, 2^m - 1 \]
and
\[ \Psi_m = \{ \psi_{m,0}, \psi_{m,1}, \ldots, \psi_{m,2^m-1} \} \]
The Haar basis of \( L_2[0,1] \) is then
\[ \bigcup_{m=0}^{\infty} \Psi_m \]

Then, one might entertain use of the Haar basis functions through order \( M \) in constructing a predictor
\[ f(X) = \beta_0 + \sum_{m=0}^{M} \sum_{j=0}^{2^m-1} \beta_{m,j} \psi_{m,j}(X) \quad (47) \]
(with the understanding that \( \psi_{0,0} = \psi \)), a form that in general allows building of functions that are constant on consecutive intervals of length \( 1/2^{M+1} \). This form can be fit by any of the various regression methods (especially involving thresholding/selection, as a typically very large number, \( 2^{M+1} \), of basis functions is employed in (47)). Large absolute values of coefficients \( \beta_{m,j} \) encode scales at which important variation in the value of the index \( m \), and location in \([0,1]\) where that variation occurs in the value \( j/2^m \). Where (perhaps after model selection/ thresholding) only a relatively few fitted coefficients are important, the corresponding scales and locations provide an informative and compact summary of the fit.

In special situations where \( N = 2^K \) and
\[ x_i = i \left( \frac{1}{2^K} \right) \quad \text{for} \quad i = 1, 2, \ldots, 2^K \]
and one uses the Haar basis functions through order \( K - 1 \), the fitting of (47) is computationally clean, since the vectors
\[ \begin{pmatrix} \psi_{m,j}(x_1) \\ \vdots \\ \psi_{m,j}(x_N) \end{pmatrix} \]
(together with the column vector of 1’s) are orthogonal. (In fact, upon division by \( N = 2^K \), they form an orthonormal basis for \( \mathbb{R}^N \).)

The Haar wavelet basis functions are easy to describe and understand. But they are discontinuous, and from some points of view that is unappealing. Other sets of wavelet basis functions have been developed that are smooth. The construction begins with a smooth "mother wavelet" in place of the step function used above. HTF make some discussion of the smooth "symmlet" wavelet basis.
5 Kernel Smoothing Methods (Ch 6 of HTF)

The central idea of this material is that when finding \( \hat{f}(x_0) \) I might weight points in the training set according to how close they are to \( x_0 \), do some kind of fitting around \( x_0 \), and ultimately read off \( \hat{f} \) from the value of the fit at \( x_0 \).

5.1 One-dimensional Kernel Smoothers

For the time being, suppose that \( X \) takes values in \([0, 1]\). Invent weighting schemes for points in the training set by defining a (usually, symmetric about 0) non-negative, real-valued function \( D(t) \) that is non-increasing for \( t \geq 0 \) and non-decreasing for \( t \leq 0 \). Often \( D(t) \) is taken to have value 0 unless \( |t| \leq 1 \). Then, a kernel function is

\[
K_\lambda(x, x_0) = D \left( \frac{x - x_0}{\lambda} \right)
\]

(48)

where \( \lambda \) is a "bandwidth" parameter that controls the rate at which weights drop off as one moves away from \( x_0 \) (and indeed in the case that \( D(t) = 0 \) for \( |t| > 1 \), how far one moves away from \( x_0 \) before no weight is assigned). Common choices for \( D \) are

1. the Epanechnikov quadratic kernel, \( D(t) = \frac{3}{4} (1 - t^2) I[|t| \leq 1] \),
2. the "tri-cube" kernel, \( D(t) = \left( 1 - |t|^3 \right)^3 I[|t| \leq 1] \), and
3. the standard normal density, \( D(t) = \phi(t) \).

Using weights (48) to make a weighted average of training responses, one arrives at the Nadaraya-Watson kernel-weighted prediction at \( x_0 \)

\[
\hat{f}_\lambda(x_0) = \frac{\sum_{i=1}^{N} K_\lambda(x_0, x_i) y_i}{\sum_{i=1}^{N} K_\lambda(x_0, x_i)}
\]

(49)

This typically smooths training outputs \( y_i \) in a more pleasing way than does a \( k \)-nearest neighbor average, but it has obvious problems at the ends of the interval \([0, 1]\) and at places in the interior of the interval where training data are dense to one side of \( x_0 \) and sparse to the other, if the target \( E[Y|X=x] \) has non-zero derivative at \( x_0 \). For example, at \( x_0 = 1 \) only \( x_i \leq 1 \) get weight, and if \( E[Y|X=x] \) is decreasing at \( x_0 = 1 \), \( \hat{f}_\lambda(1) \) will be positively biased. That is with usual symmetric kernels, (49) will fail to adequately follow an obvious trend at 0 or 1 (or at any point between where there is a sharp change in the density of input values in the training set).

A way to fix this problem with the Nadaraya-Watson predictor is to replace the locally-fitted constant with a locally-fitted line. That is, at \( x_0 \) one might choose \( \alpha(x_0) \) and \( \beta(x_0) \) to

\[
\text{minimize} \sum_{i=1}^{N} K_\lambda(x_0, x_i) (y_i - (\alpha(x_0) + \beta(x_0) x_i))^2
\]

(50)
and then employ the prediction
\[
\hat{f}_\lambda (x_0) = \alpha (x_0) + \beta (x_0) x_0
\]  
(51)

Now the weighted least squares problem (50) has an explicit solution. Let
\[
B_{N \times 2} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{pmatrix}
\]
and take
\[
W(x_0) = \text{diag}(K_\lambda (x_0, x_1), \ldots, K_\lambda (x_0, x_N))
\]
then (51) is
\[
\hat{f}_\lambda (x_0) = (1, x_0) (B'W(x_0)B)^{-1} B'W(x_0) Y
\]
\[
= \ell' (x_0) Y
\]
for the $1 \times N$ vector $\ell' (x_0) = (1, x_0) (B'W(x_0)B)^{-1} B'W(x_0)$. It is thus obvious that locally weighted linear regression is (an albeit $x_0$-dependent) linear operation on the vector of outputs. The weights in $\ell' (x_0)$ combine the original kernel values and the least squares fitting operation to produce a kind of "equivalent kernel" (for a Nadaraya-Watson type weighted average).

Recall that for smoothing splines, we represented smoothed values as
\[
\hat{Y}_\lambda = S_\lambda Y
\]
where the parameter $\lambda$ is the penalty weight, and employed
\[
df(\lambda) = \text{tr}(S_\lambda)
\]
We may do something parallel in the present context. We may take
\[
L_{\lambda N \times 2} = \begin{pmatrix} \ell' (x_1) \\ \ell' (x_2) \\ \vdots \\ \ell' (x_N) \end{pmatrix}
\]
where now the parameter $\lambda$ is the bandwidth, write
\[
\hat{Y}_\lambda = L_\lambda Y
\]
and define
\[
df(\lambda) = \text{tr}(L_\lambda)
\]
HTF suggest that matching degrees of freedom for a smoothing spline and a kernel smoother produces very similar equivalent kernels, smoothers, and predictions.
A direct generalization of 1-dimensional kernel smoothing to \( p \) dimensions might go roughly as follows. For \( D \) as before, and \( x \in \mathbb{R}^p \), I might set

\[
K_\lambda (x_0, x) = D \left( \frac{\|x - x_0\|}{\lambda} \right)
\]

and fit linear forms locally by solving the problem

\[
\min_{\alpha(x_0) \text{ and } \beta(x_0)} \sum_{i=1}^{N} K_\lambda (x_0, x_i) \left( y_i - \left( \alpha (x_0) + \beta'(x_0) x_i \right) \right)^2
\]

choosing \( \alpha (x_0) \in \mathbb{R} \) and \( \beta(x_0) \in \mathbb{R}^p \), and predicting as

\[
\hat{f}_\lambda (x_0) = \alpha (x_0) + \beta'(x_0) x_0
\]

This seems to be typically done after standardizing the coordinates of \( X \) and can be effective as long as \( N \) is not too small and \( p \) is not more than 2 or 3. However for \( p > 3 \), the curse of dimensionality comes into play and \( N \) points just usually aren’t dense enough in \( p \)-space to make direct use of kernel smoothing effective. If the method is going to be successfully used in \( \mathbb{R}^p \) it will need to be applied under appropriate structure assumptions. HTF suggest several clever versions of such assumptions in Sections 6.3 and 6.4.

5.2.1 Structured Kernels

One way to apply additional structure to the \( p \)-dimensional kernel smoothing problem is to essentially reduce input variable dimension by replacing the kernel (52) with

\[
K_{\lambda, A} (x_0, x) = D \left( \sqrt{(x - x_0)' A (x - x_0)} \right)
\]

for an appropriate non-negative definite matrix \( A \). For the eigen decomposition of \( A \),

\[
A = VDV'
\]

write

\[
(x - x_0)' A (x - x_0) = \left( D^{1/2} V' (x - x_0) \right)' \left( D^{1/2} V' (x - x_0) \right)
\]

This amounts to using not \( x \) and \( \mathbb{R}^p \) distance from \( x \) to \( x_0 \) to define weights, but rather \( D^{1/2} V' x \) and \( \mathbb{R}^p \) distance from \( D^{1/2} V' x \) to \( D^{1/2} V' x_0 \). In the event that some entries of \( D \) are 0 (or are nearly so), we basically reduce dimension from \( p \) to the number of large eigenvalues of \( A \) and define weights in a space of that dimension (spanned by eigenvectors corresponding to non-zero eigenvalues) where the curse of dimensionality may not preclude effective use of kernel smoothing. The "trick" is, of course, identifying the right directions into which to project. (Searching for such directions is part of the Friedman "projection pursuit" ideas.)
5.2.2 Structured Regression Functions

Another way to apply additional structure to the $p$-dimensional kernel smoothing problem is through assumptions on the form of the predictor fit. For example, if one assumes additivity in a predictor

$$f(X) = \alpha + \sum_{j=1}^{p} g_j(X_j)$$  \hspace{1cm} (53)

one might apply kernel smoothing in succession to the variables

$$W_j \equiv Y - \alpha + \sum_{i \neq j} \hat{g}_i(X_i)$$

as functions of the 1-dimensional variables $X_j$ in order to produce a new version of $\hat{g}_j(X_j)$, iterating until convergence to, say $\hat{f}(X)$. This might be termed a "backfitting algorithm." And where form (53) might be termed a "main effects model," the same kind of algorithm might be applied to use kernel smoothing to fit a "main effects and two factor interactions model" at some iterations doing smoothing of appropriate "residuals" as functions of 1-dimensional $X_j$ and at other iterations doing smoothing as functions of 2-dimensional $(X_j, X_j')$.

Another possibility for introducing structure assumptions and making use of low-dimensional kernel smoothing in a large $p$ situation, is by making strong global assumptions on the forms of the influence of some input variables on the output, but allowing parameters of those forms to vary in a flexible fashion with the values of some small number of coordinates of $X$. For sake of example, suppose that $p = 4$. One might consider predictors of the form

$$f(X) = \alpha (X_3, X_4) + \beta_1 (X_3, X_4) X_1 + \beta_2 (X_3, X_4) X_2$$

That is, one might assume that for fixed $(X_3, X_4)$, the form of the predictor is linear in $(X_1, X_2)$, but that the coefficients of that form may change in a flexible way with $(X_3, X_4)$. Fitting might then be approached by locally weighted least squares, with only $(X_3, X_4)$ involved in the setting of the weights. That is, one might for each $(x_{30}, x_{40})$, minimize over choices of $\alpha(x_{30}, x_{40})$, $\beta_1(x_{30}, x_{40})$ and $\beta_2(x_{30}, x_{40})$ the weighted sum of squares

$$\sum_{i=1}^{N} K_\lambda ((x_{30}, x_{40}), (x_{3i}, x_{4i})) (y_i - (\alpha(x_{30}, x_{40}) + \beta_1(x_{30}, x_{40}) x_{1i} + \beta_2(x_{30}, x_{40}) x_{2i}))^2$$

and then employ the predictor

$$\hat{f}(x_0) = \hat{\alpha}(x_{30}, x_{40}) + \hat{\beta}_1(x_{30}, x_{40}) x_{10} + \hat{\beta}_2(x_{30}, x_{40}) x_{20}$$

This kind of device keeps the dimension of the space where one is doing smoothing down to something manageable.
5.3 Kernel Density Estimation

The balance of Chapter 6 of HTF seems to be a potpourri of topics whose name might include the word "kernel" (without any necessary real connection to the "kernel smoothing" that is the chapter’s main topic). The principal one of these topics that deserves mention here is that of "kernel density estimation." The problem here is no longer smoothing to produce a fitted $f(X)$, but rather a solution to the problem:

given $X_1, X_2, \ldots, X_N$ iid with (unknown) pdf $f_X$, how to estimate $f_X$?

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

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

One may, if one then wishes, think of a corresponding "kernel"

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

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

$$\hat{f}_\lambda(x_0) = \frac{1}{N} \sum_{i=1}^{N} g(x_0|x_i, \lambda) = \frac{1}{\lambda N} \sum_{i=1}^{N} K_\lambda(x_0, x_i)$$

This is not a kernel-weighted average of outputs, but an average of kernel values.

A standard choice of univariate density $g(\cdot)$ is $\varphi(\cdot)$, the standard normal pdf. The natural generalization of this to $p$ dimensions is to let $g(\cdot)$ be the MVN$_p(\theta, \lambda^2 I)$ density. One should expect that unless $N$ is huge, this methodology will be reliable only for fairly small $p$ (say 3 at most) as a means of estimating a general $p$-dimensional pdf.

There is some discussion in HTF of classification based on estimated multivariate densities, and in particular on "naive" estimators. That is, for purposes of defining classifiers (not for the purpose of real density estimation) one might treat elements $X_j$ of $X$ as if they were independent, and multiply together kernel estimates of marginal densities. Apparently, even where the assumption of independence is clearly "wrong" this naive methodology can produce classifiers that work pretty well.
6 Understanding and Predicting Predictor Performance (Ch 7 of HTF)

There are a variety of theoretical and empirical quantities that might be computed to quantify predictor performance. Those that are empirical and reliably track important theoretical ones might potentially be used to select an effective predictor. We'll here consider some of those (theoretical and empirical) measures. We will do so in the by-now-familiar setting where training data \((X_1, Y_1), (X_2, Y_2), \ldots, (X_N, Y_N)\) are assumed to be iid \(P\) and independent of \((X, Y)\) that is also \(P\) distributed, and are used to pick a prediction rule \(f(X)\) to be used under a loss \(L(\hat{y}, y) \geq 0\).

What is very easy to think about and compute is the empirical quantity

\[
\text{err} = \frac{1}{N} \sum_{i=1}^{N} L(f(x_i), y_i)
\]

that HTF call the training error. This decreases with complexity in the form of \(f\), and is no reliable indicator of predictor performance off the training set.

Measures of prediction rule performance off the training set must have a theoretical basis (or be somehow based on data held back from the process of prediction rule development). In Section 1.4, the quantity \(E_N E[(f(x) - Y)^2 | X = x]\) was considered and called a "test error" or "generalization error" under squared error loss. (Recall that \(E_N\) was used to stand for expectation with respect to the joint distribution of the training data and \(E[\cdot|X = x]\) for conditional expectation and \(\text{Var}[Y \cdot |X = x]\) for conditional variance based on the conditional distribution of \(Y|X = x\).) A variance-bias decomposition of this was provided, one value of \(x\) at a time.

A shorthand adopted in HTF for this kind of version of test error is

\[
\text{Err}(x) = E_N E[L(f(X), Y) | X = x]
\]

In Chapter 7, HTF seem to prefer to average this kind of \(\text{Err}(x)\) according to the marginal distribution of \(X\), say \(P_X\). They first define a measure of prediction error that holds the training set fixed and averages out over the distribution of \((X, Y)\). That is, letting \(T\) stand for \(\{(X_1, Y_1), (X_2, Y_2), \ldots, (X_N, Y_N)\}\) they let

\[
\text{Err}_T = E_L(f(X), Y)
\]

and call this a "test error" or "generalization error" or "prediction error." Ideally, one would like this kind of measure to be guaranteed to be small for the training set in hand. But control of \(\text{Err}_T\) is typically not possible. One has a better chance of controlling the mean of this over the distribution of the training set,

\[
\text{Err} = E_N (\text{Err}_T)
\]

an expected prediction or test error. Notice that this is also

\[
\text{Err} = E(\text{Err}(X))
\]
A slightly different and semi-empirical version of this expected prediction error (55) is the "in-sample" test error (7.12) of HTF

\[ \frac{1}{N} \sum_{i=1}^{N} \text{Err}(x_i) \]

### 6.1 More on Predictor Squared Bias

The variance-bias decomposition of Section 1.4 is

\[ \text{Err}(x) = \text{Var}_N(f(x)) + (E_Nf(x) - E[Y|X=x])^2 + \text{Var}[Y|X=x] \]

Here consider the expected value of the squared bias (the mean of the second term of this decomposition)

\[ E(E_Nf(X) - E[Y|X])^2 \]

Suppose that the training data \( T \) are used to select a prediction rule \( f \) from some linear subspace of \( L_2(P_X) \), say \( g \), that is

\[ f = g_T \]

Notice that since subspaces are convex,

\[ g^* = E_Nf = E_Ng_T \in \{g\} \]

Further, suppose that

\[ g^* = \arg\min_{\{g\}} E(g(X) - E[Y|X])^2 \]

That is, \( g^* \) is the \( L_2(P_X) \) projection of \( E[Y|X] \) onto \( \{g\} \). Then write

\[ h^*(x) = E[Y|X=x] - g^*(x) \]

so that

\[ E[Y|X=x] = g^*(x) + h^*(x) \]

Now \( h^* \perp \{g\} \) in \( L_2(P_X) \) (and thus \( h^* \perp g^* \) and \( h^* \perp g^* \)) so that

\[ E(E_Nf(X) - E[Y|X])^2 = E(g^{**}(X) - E[Y|X])^2 \]

\[ = E(g^{**}(X) - (g^*(X) + h^*(X)))^2 \]

\[ = E(g^{**}(X) - g^*(X))^2 + Eh^*(X)^2 \]

\[ -2E(g^{**}(X) - g^*(X))h^*(X) \]

\[ = E(g^{**}(X) - g^*(X))^2 + Eh^*(X)^2 \]

The term \( Eh^*(X)^2 \) is an average squared "model bias." It is controlled by the size of the class of models fit, i.e. the size of \( \{g\} \). On the other hand, \( E(g^{**}(X) - g^*(X))^2 \) is an average squared estimation bias. This is controlled by the fitting method. One may be willing to trade an increase in it off against a decrease in \( E\text{Var}_N(f(X)) \) in the search for a good predictor.
6.2 Optimism of the Training Error

Typically, \( \overline{\text{err}} \) is less than \( \text{Err}_T \). Part of the difference in these is potentially due to the fact that \( \text{Err}_T \) is an "extra-sample" error in that the averaging in (54) is potentially over values \( x \) outside the set of values in the training data. We might consider instead

\[
\text{Err}_{T\text{in}} = \frac{1}{N} \sum_{i=1}^{N} E_Y \cdot L (Y_i^*, f(x_i))
\]

where the expectations indicated in (56) are over \( Y_i^* \sim P_{Y|X=x_i} \) (the entire training sample used to choose \( f \), both inputs and outputs, is being held constant in the averaging in (56)). The difference

\[
\text{op} = \text{Err}_{T\text{in}} - \overline{\text{err}}
\]

is called the "optimism of the training error." HTF use the notation

\[
\omega = E_Y \cdot \text{op} = E_Y (\text{Err}_{T\text{in}} - \overline{\text{err}})
\]

where the averaging indicated by \( E_Y \) is over the outputs in the training set (using the conditionally independent \( Y_i \)'s, \( Y_i \sim P_{Y|X=x_i} \)). HTF say that for many losses

\[
\omega = \frac{2}{N} \sum_{i=1}^{N} \text{Cov}_Y \left( \hat{Y}_i, Y_i \right)
\]

For example, consider the case of squared error loss. There

\[
\begin{align*}
\omega &= E_Y (\text{Err}_{T\text{in}} - \overline{\text{err}}) \\
&= E_Y E_{Y^*} \cdot \frac{1}{N} \sum_{i=1}^{N} \left( Y_i^* - f(x_i) \right)^2 - E_Y \frac{1}{N} \sum_{i=1}^{N} \left( Y_i - f(x_i) \right)^2 \\
&= \frac{2}{N} \sum_{i=1}^{N} (E_Y Y_i f(x_i) - E_Y E_{Y^*} Y_i^* f(x_i)) \\
&= \frac{2}{N} \sum_{i=1}^{N} E_Y f(x_i) (Y_i - E[Y|X=x_i]) \\
&= \frac{2}{N} \sum_{i=1}^{N} \text{Cov}_Y \left( \hat{Y}_i, Y_i \right)
\end{align*}
\]

The quantity \( \sum_{i=1}^{N} \text{Cov}_Y \left( \hat{Y}_i, Y_i \right) \) turns out to have an important connection to earlier developments. Consider linear predictors for which

\[
\hat{Y} = MY
\]
Then under a model that says that $\text{Var}_Y Y = \sigma^2 I$

$$\text{Var}_Y \begin{pmatrix} \hat{Y} \\ Y \end{pmatrix} = \sigma^2 \begin{pmatrix} M \\ I \end{pmatrix} I (M' I)$$

$$= \sigma^2 \begin{pmatrix} MM' & M' \\ M & I \end{pmatrix}$$

and in this context, the terms $\text{Cov}_Y \left( \hat{Y}_i, Y_i \right)$ are the diagonal elements of the upper right block of this covariance matrix. But that means that in the context of linear predictors

$$\sum_{i=1}^N \text{Cov}_Y \left( \hat{Y}_i, Y_i \right) = \sigma^2 \text{tr} (M)$$

For linear predictors we have called $\text{tr}(M)$ the degrees of freedom of the fit. This suggests that a plausible general definition for any fitting method producing values $\hat{Y}$ is

$$\text{d.f.} \left( \hat{Y} \right) = \frac{\sum_{i=1}^N \text{Cov}_Y \left( \hat{Y}_i, Y_i \right)}{\sigma^2}$$

### 6.3 $C_p$, AIC and BIC

The fact that

$$\text{Err}_{\text{fin}} = \text{err} + \text{op}$$

suggests the making of estimates of $\omega = E_Y \text{op}$ and the use of

$$\text{err} + \hat{\omega}$$

as a guide in model selection. This idea produces consideration of the model selection criteria $C_p$/AIC and BIC.

#### 6.3.1 $C_p$ and AIC

For the situation of least squares fitting with $p$ predictors or basis functions and squared error loss,

$$\text{d.f.} \left( \hat{Y} \right) = p = \text{tr} \left( X (X' X)^{-1} X \right)$$

so that $\sum_{i=1}^N \text{Cov}_Y \left( \hat{Y}_i, Y_i \right) = p \sigma^2$. Then, if $\hat{\sigma}^2$ is an estimated error variance based on a low-bias/high number of predictors fit, a version of (59) suitable for this context is Mallows’ $C_p$

$$C_p \equiv \text{err} + \frac{2p \hat{\sigma}^2}{N}$$
In a more general setting, if one can appropriately evaluate or estimate $\sum_{i=1}^{N} \text{Cov}_{Y} \left( \hat{Y}_{i}, Y_{i} \right) = \text{d.f.} \left( \hat{Y} \right) \sigma^{2}$, a general version of (59) becomes the Akaike information criterion

$$AIC = \ell + \frac{2}{N} \sum_{i=1}^{N} \text{Cov}_{Y} \left( \hat{Y}_{i}, Y_{i} \right) = \ell + \frac{2}{N} \text{d.f.} \left( \hat{Y} \right) \sigma^{2}$$

### 6.3.2 BIC

For situations where fitting is done by maximum likelihood, the Bayesian Information Criterion of Schwarz is an alternative to AIC. That is, where the joint distribution $P$ produces density $P \left( y_{i} | \theta, x_i \right)$ for the conditional distribution of $Y | X = x$ and $\hat{\theta}$ is the maximum likelihood estimator of $\theta$, a (maximized) log-likelihood is

$$\loglik = \sum_{i=1}^{N} \log P \left( y_{i} | \hat{\theta}, x_i \right)$$

and the so-called Bayesian information criterion

$$BIC = -2 \loglik + \left( \log N \right) \text{d.f.} \left( \hat{Y} \right)$$

For $Y | X$ normal with variance $\sigma^{2}$, up to a constant, this is

$$BIC = \frac{N}{\sigma^{2}} \left[ \ell + \frac{\left( \log N \right)}{N} \text{d.f.} \left( \hat{Y} \right) \sigma^{2} \right]$$

and after switching $2$ for $\log N$, BIC is a multiple of AIC. The replacement of $2$ with $\log N$ means that when used to guide model selections, BIC will typically favor simpler models than will AIC.

The Bayesian origins of BIC can be developed as follows. Suppose that $M$ models are under consideration, the $m$th of which has parameter vector $\theta_m$ and corresponding density for training data

$$f_{m} \left( T | \theta_{m} \right)$$

with prior density for $\theta_m$

$$g_{m} \left( \theta_{m} \right)$$

and prior probability for model $m$

$$\pi \left( m \right)$$

With this structure, the posterior distribution of the model index is

$$\pi \left( m | T \right) \propto \pi \left( m \right) \int f_{m} \left( T | \theta_{m} \right) g_{m} \left( \theta_{m} \right) d\theta_{m}$$
Under 0-1 loss and uniform $\pi (\cdot )$, one wants to choose model $m$ maximizing

$$ \int f_m (T|\theta_m) g_m (\theta_m) d\theta_m = f_m (T) = \text{the $m$th marginal of $T$} $$

Apparently, the so-called Laplace approximation says that

$$ \log f_m (T) \approx \log f_m (T|\hat{\theta}_m) - \frac{d_m}{2} \log N + O (1) $$

where $d_m$ is the real dimension of $\theta_m$. Assuming that the marginal of $X$ doesn’t change model to model or parameter to parameter, $\log f_m (T|\hat{\theta}_m)$ is loglik+$C_N$, where $C_N$ is a function of only the input values in the training set. Then

$$ -2 \log f_m (T) \approx -2 \log (\text{loglik}) + (\log N) d_m + O (1) - 2C_N $$

$$ = BIC + O (1) - 2C_N $$

and (at least approximately) choosing $m$ to maximize $f_m (T)$ is choosing $m$ to minimize BIC.

### 6.4 Cross-Validation Estimation of $\text{Err}$

This is an attempt to directly estimate

$$ \text{Err} = E_N (\text{Err}_T) = E_N EL (f (X),Y) $$

$K$-fold cross-validation consists of

1. breaking the training set into $K$ disjoint roughly equal-sized pieces, say $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_K$

2. training on each of the reduced training sets $\mathcal{T} - \mathcal{T}_k$, to produce $K$ predictors $f^k$,

3. letting $k (i)$ be the piece $\mathcal{T}_k$ containing training case $i$, and computing the cross-validation error

$$ CV (f) = \frac{1}{N} \sum_{i=1}^{N} L \left( y_i, f^{k(i)} (x_i) \right) $$

This is roughly the same as fitting on each $\mathcal{T} - \mathcal{T}_k$ and correspondingly evaluating on $\mathcal{T}_k$, and then averaging. (When $N$ is a multiple of $K$, these are exactly the same.) One hopes that $CV (f)$ approximates $\text{Err}$.

The choice of $K = N$ is called "leave one out cross-validation" and sometimes in this case there are slick computational ways of evaluating $CV (f)$.

In model selection, say where predictor $f$ has a tuning parameter $\alpha$, we look at

$$ CV (f_\alpha) $$
as a function of $\alpha$, try to optimize, and then refit (with that $\alpha$) to the whole training set.

$K$-fold cross-validation can be expected to estimate

$$ \text{Err} = E_{\cdot N} E L \left( f \left( X \right), Y \right) $$

for

$$ "N" = \left( 1 - \frac{1}{K} \right) N $$

The question of how cross-validation might be expected to do is thus related to how Err changes with $N$ (the size of the training sample). Typically, Err decreases monotonically in $N$ approaching some limiting value as $N$ goes to infinity. The "early" (small $N$) part of the "ERR vs. $N$ curve" is steep and the "late" part (large $N$) is relatively flat. If $(1 - 1/K) N$ is large enough that at such size of the training data set, the curve is flat, then the effectiveness of cross-validation is limited only by the noise inherent in estimating it, and not by the fact that training sets of size $(1 - 1/K) N$ are not of size $N$. Operationally, $K = 5$ or 10 seem standard, and a common rule of thumb is to select for use the predictor of minimum complexity whose cross-validation error is within one standard error of the minimum $CV(f)$ for predictors $f$ under consideration. Presumably, "standard error" here refers to $(K/N)^{-1/2}$ times a sample standard deviation of the $K$ values

$$ \frac{K}{N} \sum_{i \in k(i) = k} L \left( y_i, f^k \left( x_i \right) \right) $$

(though it’s not completely obvious which complexity’s standard error is under discussion when one says this).

HTF say that for many linear fitting methods (that produce $\hat{Y} = M Y$) including least squares projection and cubic smoothing splines, the $N = K$ (leave one out) cross-validation error is for $f^i$ produced by training on $T - \{i\}$

$$ CV \left( f \right) = \frac{1}{N} \sum_{i=1}^{N} \left( y_i - f^i \left( x_i \right) \right)^2 = \frac{1}{N} \sum_{i=1}^{N} \left( \frac{y_i - f \left( x_i \right)}{1 - M_{ii}} \right)^2 $$

(for $M_{ii}$ the $i$th diagonal element of $M$.) The so-called generalized cross-validation approximation to this is the much more easily computed

$$ GCV \left( f \right) = \frac{1}{N} \sum_{i=1}^{N} \left( \frac{y_i - f \left( x_i \right)}{1 - \text{tr} \left( M \right) / N} \right)^2 $$

$$ = \frac{\text{Err}}{(1 - \text{tr} \left( M \right) / N)^2} $$

54
It is worth noting (per HTF Exercise 7.7) that since \(1/(1-x)^2 \approx 1+2x\) for \(x\) near 0,

\[
GCV(f) = \frac{1}{N} \sum_{i=1}^{N} \left( \frac{y_i - f(x_i)}{1 - \text{tr}(M)/N} \right)^2
\]

\[
\approx \frac{1}{N} \sum_{i=1}^{N} (y_i - f(x_i))^2 + \frac{2}{N} \text{tr}(M) \left( \frac{1}{N} \sum_{i=1}^{N} (y_i - f(x_i))^2 \right)
\]

which is close to AIC, the difference being that here \(\sigma^2\) is being estimated based on the model being fit, as opposed to being estimated based on a low-bias/large model.

### 6.5 Bootstrap Estimation of Err

Suppose that the values of the input vectors in the training set are unique. One might make \(B\) bootstrap samples of \(N\) (random samples with replacement of size \(N\)) from the training set \(\mathcal{T}\), say \(\mathcal{T}_1^*, \mathcal{T}_2^*, \ldots, \mathcal{T}_B^*\), and train on these bootstrap samples to produce predictors, say

\[
\text{predictor } f^{*b} \text{ based on } \mathcal{T}_b^*
\]

Let \(C^i\) be the set of indices \(b = 1, 2, \ldots, B\) for which \((x_i, y_i) \notin \mathcal{T}_b^*\). A possible bootstrap estimate of Err is then

\[
\hat{\text{Err}}^{(1)} = \frac{1}{N} \sum_{i=1}^{N} \left[ \frac{1}{\#C^i} \sum_{b \in C^i} L(y_i, f^{*b}(x_i)) \right]
\]

It’s not completely clear what to make of this. For one thing, the \(\mathcal{T}_b^*\) do not have \(N\) distinct elements. In fact, it’s fairly easy to see that the expected number of distinct cases in a bootstrap sample is about \(0.632N\). That is, note that

\[
P[x_1 \text{ is in a particular bootstrap sample } \mathcal{T}^*] = 1 - \left(1 - \frac{1}{N}\right)^N \rightarrow 1 - \exp(-1) = 0.632
\]

Then

\[
\mathbb{E} \left( \text{number of distinct cases in } \mathcal{T}^* \right) = \mathbb{E} \left( \sum_{i=1}^{N} I[x_i \text{ is in } \mathcal{T}^*] \right)
\]

\[
= \sum_{i=1}^{N} \mathbb{E} I[x_i \text{ is in } \mathcal{T}^*]
\]

\[
\approx \sum_{i=1}^{N} 0.632
\]

\[= 0.632N\]
So roughly speaking, we might expect $\hat{\text{Err}}^{(1)}$ to estimate $\text{Err}$ at $.632N$, not at $N$. So unless $\text{Err}$ as a function of training set size is fairly flat to the right of $.632N$, one might expect substantial positive bias in it as an estimate of $\text{Err}$ (at $N$).

HTF argue for

$$\hat{\text{Err}}^{(.632)} \equiv .368\hat{\text{Err}}^{(1)} + .632\hat{\text{Err}}^{(1)}$$

as a first order correction on the biased bootstrap estimate, but admit that this is not perfect either, and propose a more complicated fix (that they call $\hat{\text{Err}}^{(.632+)}$) for classification problems. It is hard to see what real advantage the bootstrap has over $K$-fold cross validation, especially since one will typically want $B$ to be substantially larger than usual values for $K$.

7 Inference and Model Averaging (Ch 8 of HTF)

Much of Chapter 8 of HTF is generalities about ordinary statistical inference common in any Statistics graduate program. We’ll not say much about them, both because they are already in your background, and also because they are really off the basic machine learning path of prediction. One part of this that does perhaps merit a bit of exposition is Bayes development of predictors. The balance of the chapter is then about "model averaging."

7.1 Bayes Predictors

Consider first a very simple context where $P$ has parameter $\theta$ and therefore the conditional mean of the output is a parametric (in $\theta$) function of $x$

$$E_{\theta} [Y|X = x] = m(x|\theta)$$

Suppose that the conditional distribution of $Y|X = x$ has density $f_{\theta}(y|x)$. Then for a prior distribution on $\theta$ with density $g(\theta)$, the posterior has density

$$g(\theta|T) \propto \prod_{i=1}^{N} f_{\theta}(y_{i}|x_{i}) g(\theta)$$

For any fixed $x$, this posterior distribution induces a corresponding distribution on $m(x|\theta)$. Then, a sensible predictor based on the Bayes structure is

$$f(x) = E[m(x|\theta)|T] = \int m(x|\theta) g(\theta|T) d\theta$$

A second, more specific and complicated application of essentially Bayesian thinking to the development of a predictor might be based the use of a Gaussian process as a more or less non-parametric "prior distribution" for the function $E[Y|X = x]$. That is, for purposes of developing a predictor, suppose that one assumes that

$$y = \eta(x) + \epsilon$$
where

\[ \eta(x) = \mu(x) + \gamma(x) \]

\[ \mathbb{E}\epsilon = 0, \ Var\epsilon = \sigma^2 \]

the function \( \mu(x) \) is known (it could be taken to be identically 0) and plays the role of a prior mean for

\[ \eta(x) = \mathbb{E}[Y | X = x] \]

and (independent of the errors \( \epsilon \)), \( \gamma(x) \) is a realization of a mean 0 stationary Gaussian process on \( \mathbb{R}^p \), this Gaussian process describing the prior uncertainty for \( \gamma(x) \) around \( \mu(x) \). More explicitly, the assumption on \( \gamma(x) \) is that \( \mathbb{E}\gamma(x) = 0 \) and \( \text{Var}\gamma(x) = \tau^2 \) for all \( x \), and for some appropriate (correlation) function \( \rho \), \( \text{Cov}(\gamma(x), \gamma(x')) = \tau^2 \rho(x - x') \) for all \( x \) and \( x' \) \( (\rho(0) = 1 \) and the function of two variables \( \rho(x - x') \) must be positive definite). The "Gaussian" part of the assumptions is then that we assume that for any finite set of element of \( \mathbb{R}^p \), say \( z_1, z_2, \ldots, z_M \), the corresponding values \( \gamma(z_i) \) are multivariate normal.

There are a number of standard forms that have been suggested for the correlation function \( \rho \). The simplest ones are of a product form, i.e. if \( j \) is a valid one-dimensional correlation function, then the product

\[ \rho(x - x') = \prod_{j=1}^{p} \rho_j(x_j - x'_j) \]

is a valid correlation function for a Gaussian process on \( \mathbb{R}^p \). Standard forms for correlation functions in one dimension are \( \rho(\Delta) = \exp(-c\Delta^2) \) and \( \rho(\Delta) = \exp(-c|\Delta|) \). The first produces "smoother" realizations than does the second, and in both cases, the constant \( c \) governs how fast realizations vary.

One may then consider the joint distribution (conditional on the \( x_i \) and assuming that for the training values \( y_i \) the \( \epsilon_i \) are iid independent of the \( \gamma(x_i) \)) of the training output values and a value of \( \eta(x) \). From this, one can find the conditional mean for \( \eta(x) \) given the training data. To that end, let

\[ \Sigma_{N \times N} = (\tau^2 \rho(x_i - x_j))_{i=1,2,\ldots,N}^{j=1,2,\ldots,N} \]

Then for a single value of \( x \),

\[
\begin{pmatrix}
  y_1 \\
  y_2 \\
  \vdots \\
  y_N \\
  \eta(x)
\end{pmatrix}
\sim \text{MVN}_{N+1}
\begin{pmatrix}
  \mu(x_1) \\
  \mu(x_2) \\
  \vdots \\
  \mu(x_N) \\
  \mu(x)
\end{pmatrix},
\begin{pmatrix}
  (\Sigma + \sigma^2 I) \\
  \Sigma(x)
\end{pmatrix}^T
\]

for

\[
\Sigma(x)_{N \times 1} = \begin{pmatrix}
  \tau^2 \rho(x - x_1) \\
  \tau^2 \rho(x - x_2) \\
  \vdots \\
  \tau^2 \rho(x - x_N)
\end{pmatrix}
\]

57
Then standard multivariate normal theory says that the conditional mean of \( \eta(x) \) given \( Y \) is

\[
f(x) = \mu(x) + \Sigma(x)' \left( \Sigma + \sigma^2 I \right)^{-1} \begin{pmatrix} y_1 - \mu(x_1) \\ y_2 - \mu(x_2) \\ \vdots \\ y_N - \mu(x_N) \end{pmatrix}
\]

Write

\[
w_{N \times 1} = \left( \Sigma + \sigma^2 I \right)^{-1} \begin{pmatrix} y_1 - \mu(x_1) \\ y_2 - \mu(x_2) \\ \vdots \\ y_N - \mu(x_N) \end{pmatrix}
\]

and then note that (60) implies that

\[
f(x) = \mu(x) + \sum_{i=1}^{N} w_i \tau^2 \rho(x - x_i)
\]

and we see that this development ultimately produces \( \mu(x) \) plus a linear combination of the "basis functions" \( \tau^2 \rho(x - x_i) \) as a predictor. Remembering that \( \tau^2 \rho(x - x') \) must be positive definite and seeing the ultimate form of the predictor, we are reminded of the RKHS material in Section 4.4.

In fact, consider the case where \( \mu(x) \equiv 0 \). (If one has some non-zero prior mean for \( \eta(x) \), arguably that mean function should be subtracted from the raw training outputs before beginning the development of a predictor. At a minimum, output values should probably be centered before attempting development of a predictor.) Compare (61) and (62) to (44) and (45) for the \( \mu(x) = 0 \) case.

What is then clear is that the present "Bayes" Gaussian process development of a predictor under squared error loss based on a covariance function \( \tau^2 \rho(x - x') \) and error variance \( \sigma^2 \) is equivalent to a RKHS regularized fit of a function to training data based on a kernel \( K(x, x') = \tau^2 \rho(x - x') \) and penalty weight \( \lambda = \sigma^2 \).

### 7.2 Predictors Built on Bootstrap Samples

One might make \( B \) bootstrap samples of \( N \) (random samples with replacement of size \( N \)) from the training set \( T \), say \( T_1^*, T_2^*, \ldots, T_B^* \), and train on these bootstrap samples to produce, say,

\[
\text{predictor } f^{*b} \text{ based on } T^*_b
\]

Now (rather than using these to estimate the prediction error as in Section 6.5) consider a couple of ways of using these to build a predictor (under squared error loss).
7.2.1 Bagging

The possibility considered in Section 8.7 of HTF is "bootstrap aggregation," or "bagging." This is use of the predictor

\[ f_{\text{bag}}(x) \equiv \frac{1}{B} \sum_{b=1}^{B} f^{*b}(x) \]

Notice that even for fixed training set \( T \) and input \( x \), this is random (varying with the selection of the bootstrap samples). One might let \( E_\ast \) denote averaging over the creation of a single bootstrap sample and \( f_\ast \) be the predictor derived from such a bootstrap sample and think of

\[ E_\ast f_\ast(x) \]

as the "true" bagging predictor (that has the simulation-based approximation \( f_{\text{bag}}(x) \)). One is counting on a law of large numbers to conclude that \( f_{\text{bag}}(x) \rightarrow E_\ast f_\ast(x) \) as \( B \rightarrow \infty \). Note too, that unless the operations applied to a training set to produce \( f \) are linear, \( E_\ast f_\ast(x) \) will differ from the predictor computed from the training data, \( f(x) \).

Where a loss other than squared error is involved, exactly how to "bag" bootstrapped versions of a predictor is not altogether obvious, and apparently even what might look like sensible possibilities can do poorly.

7.2.2 Bumping

Another/different thing one might do with bootstrap versions of a predictor is to "pick a winner" based on performance on the training data. This is the "bumping"/stochastic perturbation idea of HTF's Section 8.9. That is, let \( f^{*0} = f \) be the predictor computed from the training data, and define

\[ \hat{b} = \arg \min_{b=0,1,\ldots,B} \sum_{i=1}^{N} (y_i - f^{*b}(x_i))^2 \]

and take

\[ f_{\text{bump}}(x) = f^{*\hat{b}}(x) \]

The idea here is that if a few cases in the training data are responsible for making a basically good predictor perform poorly, eventually a bootstrap sample will miss those cases and produce an effective predictor.

7.3 Model Averaging and Stacking

The bagging idea averages versions of a single predictor computed from different bootstrap samples. An alternative might be to somehow weight together different predictors (potentially even based on different models).
7.3.1 Bayesian Model Averaging

One theoretically straightforward way to justify this kind of enterprise and identify candidate predictors is through the Bayes "multiple model" scenario of Section 6.3.2. That is, again suppose that $M$ models are under consideration, the $m$th of which has parameter vector $\theta_m$ and corresponding density for training data

$$f_m(T|\theta_m)$$

with prior density for $\theta_m$

$$g_m(\theta_m)$$

and prior probability for model $m$

$$\pi(m)$$

As before, the posterior distribution of the model index is specified by

$$\pi(m|T) = \frac{\pi(m) \int f_m(T|\theta_m) g_m(\theta_m) d\theta_m}{\sum_{m=1}^{M} \pi(m) \int f_m(T|\theta_m) g_m(\theta_m) d\theta_m}$$

Now suppose that for each model there is a quantity $\gamma_m(\theta_m)$ that is of interest and having a common interpretation across models. Notice that under the current assumption that $P$ depends upon the parameter $\theta_m$ in model $m$, $\gamma_m(\theta_m)$ could be the conditional mean of $Y$ given $X = x$. That is, one important instance of this structure is $\gamma_m(\theta_m, x) = E_\theta[Y|X = x]$. Since in model $m$ the density of $\theta_m$ conditional on $T$ is

$$g(\theta_m|T) = \frac{f_m(T|\theta_m) g_m(\theta_m)}{\int f_m(T|\theta_m) g_m(\theta_m) d\theta_m}$$

the conditional mean of $\gamma_m(\theta_m)$ (in model $m$) is

$$E[\gamma_m(\theta_m)|m, T] = \frac{\int \gamma_m(\theta_m) f_m(T|\theta_m) g_m(\theta_m) d\theta_m}{\int f_m(T|\theta_m) g_m(\theta_m) d\theta_m}$$

and a sensible predictor of $\gamma_m(\theta_m)$ is

$$E[\gamma_m(\theta_m)|T] = \sum_{m=1}^{M} \pi(m|T) E[\gamma_m(\theta_m)|m, T]$$

a (model posterior probability) weighted average of the predictors $E[\gamma_m(\theta_m)|m, T]$ individually appropriate under the $M$ different models. In the important special case where $\gamma_m(\theta_m, x) = E_\theta[Y|X = x]$, this is

$$E[\gamma_m(\theta_m, x)|T] = \sum_{m=1}^{M} \pi(m|T) E[E_\theta[Y|X = x]|m, T]$$
7.3.2 Stacking

What might be suggested like this, but with a less Bayesian flavor? Suppose that \( M \) predictors are available (all based on the same training data), \( f_1, f_2, \ldots, f_M \). Under squared error loss, I might seek a weight vector \( \mathbf{w} \) for which the predictor

\[
f(\mathbf{x}) = \sum_{m=1}^{M} w_m f_m(\mathbf{x})
\]

is effective. I might even let \( \mathbf{w} \) depend upon \( \mathbf{x} \), producing

\[
f(\mathbf{x}) = \sum_{m=1}^{M} w_m(\mathbf{x}) f_m(\mathbf{x})
\]

HTF pose the problem of perhaps finding

\[
\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \mathbb{E}_{\mathbf{Y}} \left( Y - \sum_{m=1}^{M} w_m f_m(\mathbf{X}) \right)^2
\]

or even

\[
\hat{\mathbf{w}}(\mathbf{x}) = \arg \min_{\mathbf{w}} \mathbb{E}_{\mathbf{Y}} \left[ \left( Y - \sum_{m=1}^{M} w_m f_m(\mathbf{x}) \right)^2 | \mathbf{X} = \mathbf{x} \right]
\]

Using either the \( P \times P \) joint distribution of \((\mathbf{T}, (\mathbf{X}, Y))\) or the \( P \times P \mathbf{Y} | \mathbf{X} = \mathbf{x} \) conditional distribution of \((\mathbf{T}, Y)\) given \( \mathbf{X} = \mathbf{x} \), one may consider the random vector \((f_1(\mathbf{X}), f_2(\mathbf{X}), \ldots, f_M(\mathbf{X}), Y)'\) or the random vector \((f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_M(\mathbf{x}), Y)'\). Let \((\mathbf{f}', \mathbf{Y})\) stand for either one and

\[
\mathbb{E} \left( \begin{array}{c} \mathbf{f} \\ \mathbf{Y} \end{array} \right)_{(M+1) \times 1} = \left( \begin{array}{c} \mathbb{E} \mathbf{f} \\ \mathbb{E} \mathbf{Y} \end{array} \right)
\]

be the corresponding mean vector and

\[
\mathbb{E} (\mathbf{f} \mathbf{f}')(M \times M)
\]

be the matrix of expected products of the predictions. Upon writing out the expected square to be minimized and doing some matrix calculus, it’s possible to see that optimal weights are of the form

\[
\hat{\mathbf{w}} = (\mathbb{E} \mathbf{Y}) (\mathbb{E} (\mathbf{f} \mathbf{f}'))^{-1} \mathbb{E} \mathbf{f} \quad \text{(or } \hat{\mathbf{w}}(\mathbf{x}) \text{ is of this form)}
\]

(We observe that HTF’s notation in (8.57) could be interpreted to incorrectly place the inverse sign inside the expectation. They call this "the population linear regression of \( Y \) on \( \mathbf{f} \).") Of course, none of this is usable in practice,
as typically the mean vector and expected cross product matrix are unknown. And apparently, the simplest empirical approximations to these do not work well.

What, instead, HTF seem to recommend is to define

$$w^{\text{stack}} = \arg \min_w \sum_{i=1}^N \left( y_i - \sum_{m=1}^M w_m f_m^i (x_i) \right)^2$$

for $f_m^i$ the $m$th predictor fit to $T - \{(x_i, y_i)\}$, the training set with the $i$th case removed. Then ultimately the "stacked" predictor of $Y$ is

$$f(x) = \sum_{m=1}^M w_m^{\text{stack}} f_m(x)$$

8 Additive Models, Trees, and Related Methods (Ch 9 of HTF)

8.1 Additive Models

Section 9.1 of HTF is the book’s formal discussion of additive models. The authors have already alluded to them repeatedly. The main points of the section are:

1. Additive models are highly flexible. One may
   (a) predict continuous outputs $Y$ using squared error loss, and 0-1 classification variables using a negative log-likelihood loss and
   $$g(P[Y = 1|X = x]) = \text{an additive form}$$
   i.e.
   $$P \left[ Y = 1|X = \min_{i=1,2,...,N} x_{1i} \leq x_1 \leq \max_{i=1,2,...,N} x_{1i} \right] = g^{-1} \text{ (an additive form)}$$
   for some known link function $g$,
   (b) mix types of predictors (continuous, categorical) and types of functions of them in the additive form (see page 297) to produce all sorts of interesting models (including semi-parametric ones and ones with low order interactions).

2. Fitting is possible via the back-fitting algorithm. That is, if I am to fit (under SEL)
   $$f(x) = \alpha + \sum_{l=1}^L f_l (x^l)$$
   for $x^l$ some part of $x$, I might
• set $\hat{\alpha} = \frac{1}{N} \sum_{i=1}^{N} y_i$,
• cycle through $l = 1, 2, \ldots, L, 1, 2, \ldots$
  – fitting via some appropriate (often linear) operation
    \[ f_l(x^l) \] to "data" \{ \{(x_i^l, y_i^l)\}_{i=1,2,\ldots,N} \}
    
    for
    \[ y_i^l = y_i - \left( \hat{\alpha} + \sum_{m \neq l} f_m(x_i^m) \right) \]

    where the $f_m$ are the current versions of the fitted summands,
    – setting
    \[ f_l = \text{the newly fitted version—}\text{the sample mean of this newly fitted version across all } x_i^l \]

    (in theory this is not necessary, but it is here to prevent numerical/round-off errors from causing the $f_m$ to drift up and down by additive constants summing to 0 across $m$),

    until convergence.

There is a logistic regression version of this on page 300 of HTF.

3. This material may not be all that helpful for large $p$, as there is nothing here that does any thresholding or automatic variable selection.

### 8.2 Tree-Based Methods

This is something genuinely new to our discussion. We’ll first consider the SEL/regression version. This begins with a forward-selection/"greedy" algorithm for inventing predictions constant on $p$-dimensional rectangles, by successively looking for an optimal binary split of a single one of an existing set of rectangles. For example, with $p = 2$, one begins with a rectangle defined by

\[ a = \min_{i=1,2,\ldots,N} x_{1i} \leq x_1 \leq \max_{i=1,2,\ldots,N} x_{1i} = b \]

and

\[ c = \min_{i=1,2,\ldots,N} x_{2i} \leq x_2 \leq \max_{i=1,2,\ldots,N} x_{2i} = d \]

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 the training error

\[ SSE = \sum_{\text{rectangles } i \text{ with } x_{1i} \text{ in the rectangle}} \sum_{\text{the rectangle}} (y_i - \overline{y}_{\text{rectangle}})^2 \]

One then splits (optimally) one of the (now) two rectangles at $x_1 = s_2$ or $x_2 = s_2$, etc.
Where \( l \) rectangles \( R_1, R_2, \ldots, R_l \) in \( \mathbb{R}^p \) have been created, and
\[
m(x) = \text{the index of the rectangle to which } x \text{ belongs}
\]
the corresponding predictor is
\[
f_l(x) = \frac{1}{\# \text{ training input vectors } x_i \text{ in } R_{m(x)}} \sum_{x_i \text{ in } R_{m(x)}} y_i
\]
and in this notation the training error is
\[
SSE = \sum_{i=1}^{N} (y_i - f_l(x_i))^2
\]
If one is to continue beyond \( l \) rectangles, one then looks for a value \( s_l \) to split one of the existing rectangles \( R_1, R_2, \ldots, R_l \) on \( x_1 \) or \( x_2 \) or \( \ldots \) or \( x_p \) and thereby produce the greatest reduction in \( SSE \). (We note that there is no guarantee that after \( l \) splits one will have the best (in terms of \( SSE \)) possible set of \( l + 1 \) rectangles.)

Any series of binary splits of rectangles can be represented graphically as a binary tree, each split represented by a node where there is a fork and each final rectangle by an end node. It is convenient to discuss rectangle-splitting and "unsplitting" in terms of operations on corresponding binary trees, and henceforth we adopt this language.

It is, of course, possible to continue splitting rectangles/adding branches to a tree until every distinct \( x \) in the training set has its own rectangle. But that is not helpful in a practical sense, in that it corresponds to a very low bias/high variance predictor. So how does one find a tree of appropriate size? The standard recommendation seems to be this:

1. Grow a tree (on a given data set) until the cell with the fewest training \( x_i \) contains 5 or less such points, then
2. "prune" the tree in 1. back by for each \( \alpha > 0 \) (a complexity parameter, weighting \( SSE \) against complexity defined in terms of tree size) minimizing over choices of sub-trees, the quantity
\[
C_\alpha (T) = |T| + \alpha \cdot SSE(T)
\]
(for, in the obvious way, \( |T| \) the number of final nodes in the candidate tree and \( SSE(T) \) the error sum of squares for the corresponding predictor),
3. pick a good value of \( \alpha \) (the smallest one that has average performance "close" to the best \( SSE \) by cross-validation, call this \( \alpha_\text{opt} \), and
4. then operating on the entire training set, apply 1. and find the sub-tree, say \( T_{\alpha_\text{opt}} \) optimizing \( C_{\alpha_\text{opt}} (T) \) and use it.
The question of how to execute short of an exhaustive search over sub-trees for every different value of \( \alpha \) has a workable answer. There is a relatively small number of nested candidate sub-trees of an original tree that are the only ones that are possible minimizers of \( C_\alpha(T) \), and as \( \alpha \) decreases one moves through that nested sequence of sub-trees from the largest/original tree to the smallest. Suppose that one begins with a large tree, \( T_0 \) (with \( |T_0| \) final nodes). One may quickly search over all "pruned" versions of \( T_0 \) (sub-trees \( T \) created by removing a node where there is a fork and all branches that follow below it) and find the one with minimum

\[
\frac{SSE(T) - SSE(T_0)}{|T_0| - |T|}
\]

(This IS the per node (of the lopped off branch of the first tree) increase in \( SSE \).) Call that sub-tree \( T_1 \). \( T_0 \) is the optimizer of \( C_\alpha(T) \) over sub-trees of \( T_0 \) for every \( \alpha \geq (SSE(T_1) - SSE(T_0)) / (|T_0| - |T_1|) \), but at that value of \( \alpha \), the optimizing sub-tree switches to \( T_1 \). One then may search over all "pruned" versions of \( T_1 \) for the one with minimum

\[
\frac{SSE(T) - SSE(T_1)}{|T_1| - |T|}
\]

and call it \( T_2 \). \( T_1 \) is the optimizer of of \( C_\alpha(T) \) over sub-trees of \( T_0 \) for every \( (SSE(T_2) - SSE(T_1)) / (|T_1| - |T_2|) \leq \alpha \leq (SSE(T_1) - SSE(T_0)) / (|T_0| - |T_1|) \). But at \( \alpha = (SSE(T_2) - SSE(T_1)) / (|T_1| - |T_2|) \) the optimizing sub-tree switches to \( T_2 \), and so on. (In this process, if there ever happens to be a tie among sub-trees in terms of a minimizing a ratio of increase in error sum of squares per decrease in number of nodes, one chooses the sub-tree with the smaller \( |T| \).

For \( T_\alpha \) optimizing \( C_\alpha(T) \), the function of \( \alpha \),

\[
C_\alpha(T_\alpha) = \min_T C_\alpha(T)
\]

is piecewise linear in \( \alpha \), and both it and the optimizing nested sequence of sub-trees can be computed very efficiently in this fashion.

The continuous \( Y \), SEL version of this stuff is known as the making of "regression trees." The "classification trees" version of it is very similar. One needs only to define an empirical loss to associate with a given tree parallel to \( SSE \) used above. To that end, we first note that in a \( K \) class problem (where \( Y \) takes values in \( G = \{1, 2, \ldots, K\} \)) corresponding to a particular rectangle \( R_m \) is the fraction of training vectors with classification \( k \),

\[
\hat{p}_{mk} = \frac{1}{\# \text{ training input vectors in } R_m} \sum_{i \text{ with } x_i \text{ in } R_m} I[y_i = k]
\]

and a plausible \( G \)-valued predictor based on \( l \) rectangles is

\[
f_l(x) = \arg\max_{k \in G} \hat{p}_{mk}(x)
\]
the class that is most heavily represented in the rectangle to which \( x \) belongs. The empirical mis-classification rate for this predictor is

\[
\hat{\text{err}} = \frac{1}{N} \sum_{i=1}^{N} I[y^*_i \neq f_l(x_i)] = \frac{1}{N} \sum_{m=1}^{l} N_m (1 - \hat{p}_{mk(m)})
\]

where \( N_m = \# \) training input vectors in \( R_m \), and \( k(m) = \arg \max_{k \in \mathcal{G}} \hat{p}_{mk} \). Two related alternative forms for a training error are "the Gini index"

\[
\hat{\text{err}} = \frac{1}{N} \sum_{m=1}^{l} N_m \left( \sum_{k=1}^{K} \hat{p}_{mk} (1 - \hat{p}_{mk}) \right)
\]

and the so-called "cross entropy"

\[
\hat{\text{err}} = -\frac{1}{N} \sum_{m=1}^{l} N_m \left( \sum_{k=1}^{K} \hat{p}_{mk} \ln (\hat{p}_{mk}) \right)
\]

Upon adopting one of these forms for \( \hat{\text{err}} \) and using it to replace \( SSE \) in the regression tree discussion, one has a classification tree methodology. HTF suggest using the Gini index or cross entropy for tree growing and any of the indices (but most typically the empirical mis-classification rate for tree pruning according to cost-complexity.

8.3 PRIM (Patient Rule Induction Method)

This is another rectangle-based method of making a predictor on \( \mathbb{R}^p \). It could be termed a type of "bump-hunting." For a series of rectangles (or boxes) in \( p \)-space

\( R_1, R_2, \ldots, R_l \)

one defines a predictor

\[
f_l(x) = \begin{cases} 
\overline{y}_{R_1} & \text{if } x \in R_1 \\
\overline{y}_{R_2 - R_1} & \text{if } x \in R_2 - R_1 \\
\vdots & \vdots \\
\overline{y}_{R_m - \bigcup_{k=1}^{m-1} R_k} & \text{if } x \in R_m - \bigcup_{k=1}^{m-1} R_k \\
\vdots & \vdots \\
\overline{y}_{(\bigcup_{k=1}^{l} R_k)^c} & \text{if } x \notin \bigcup_{k=1}^{l} R_k 
\end{cases}
\]

The boxes or rectangles are defined recursively in a way intended to catch "the remaining part of the input space with the largest output values." That is, to find \( R_1 \)
1. identify a rectangle

\[
\begin{align*}
  l_1 & \leq x_1 \leq u_1 \\
  l_2 & \leq x_2 \leq u_2 \\
  \vdots \\
  l_p & \leq x_p \leq u_p
\end{align*}
\]

that includes all input vectors in the training set,

2. identify a dimension, \( j \), and either \( l_j \) or \( u_j \) so that by reducing \( u_j \) or increasing \( l_j \) just enough to remove a fraction \( \alpha \) (say \( \alpha = .1 \)) of the training vectors currently in the rectangle, the largest value of \( \overline{y}_{\text{rectangle}} \) possible is produced, and update that boundary of the rectangle,

3. repeat 2. until some minimum number of training inputs \( x_i \) remain in the rectangle (say, at least 10),

4. expand the rectangle in any direction (increase a \( u \) or decrease an \( l \)) adding a training input vector that provides a maximal increase in \( \overline{y}_{\text{rectangle}} \), and

5. repeat 4. until no increase is possible by adding a single training input vector.

This produces \( R_1 \). For what it is worth, step 2. is called "peeling" and step 4. is called "pasting."

Upon producing \( R_1 \), one removes from consideration all training vectors with \( x_j \in R_1 \) and repeats 1. through 5. to produce \( R_2 \). This continues until a desired number of rectangles has been created. One may pick an appropriate number of rectangles (\( l \) is a tuning parameter) by cross-validation and then apply the procedure to the whole training set to produce a set of rectangles and predictor on \( p \)-space that is piece-wise constant on regions built from boolean operations on rectangles.

8.4 MARS (Multivariate Adaptive Regression Splines)

This is a high-dimensional regression methodology based on the use of "hockey-stick functions" and their products as basis functions. That is, with input space \( \mathbb{R}^p \) we consider basis functions built on the \( Np \) pairs of functions

\[
h_{ij1} (\mathbf{x}) = (x_j - x_{ij})_+ \quad \text{and} \quad h_{ij2} (\mathbf{x}) = (x_{ij} - x_j)_+ \quad (63)
\]

\( (x_{ij} \) is the \( j \)th coordinate of the \( i \)th input training vector and both \( h_{ij1}(\mathbf{x}) \) and \( h_{ij2}(\mathbf{x}) \) depend on \( \mathbf{x} \) only through the \( j \)th coordinate of \( \mathbf{x} \). MARS builds predictors sequentially, making use of these "reflected pairs" roughly as follows.
1. Begin with
\[ f_0(x) = \beta_0 \]

2. Identify a pair (63) so that
\[ \beta_0 + \beta_{11} h_{ij1}(x) + \beta_{12} h_{ij1}(x) \]
has the best \( SSE \) possible. Call the selected functions
\[ g_{11} = h_{ij1} \text{ and } g_{12} = h_{ij2} \]
and set
\[ f_1(x) = \beta_0 + \beta_{11} g_{11}(x) + \beta_{12} g_{12}(x) \]

3. At stage \( l \) of the predictor-building process, with predictor
\[ f_{l-1}(x) = \beta_0 + \sum_{m=1}^{l-1} (\beta_{m1} g_{m1}(x) + \beta_{m2} g_{m2}(x)) \]
in hand, consider for addition to the model, pairs of basis functions that are either of the basic form (63) or of the form
\[ h_{ij1}(x) g_{m1}(x) \text{ and } h_{ij2}(x) g_{m1}(x) \]
or of the form
\[ h_{ij1}(x) g_{m2}(x) \text{ and } h_{ij2}(x) g_{m2}(x) \]
for some \( m < l \), subject to the constraints that no \( x_j \) appears in any candidate product more than once (maintaining the piece-wise linearity of sections of the predictor). Additionally, one may decide to put an upper limit on the order of the products considered for inclusion in the predictor. The best candidate pair in terms of reducing \( SSE \) gets called, say, \( g_{l1} \) and \( g_{l2} \) and one sets
\[ f_l(x) = \beta_0 + \sum_{m=1}^{l} (\beta_{m1} g_{m1}(x) + \beta_{m2} g_{m2}(x)) \]

One might pick the tuning parameter \( l \) by cross-validation, but the standard implementation of MARS apparently uses instead a kind of generalized cross validation error
\[ GCV(l) = \frac{\sum_{i=1}^{N} (y_i - f_l(x_i))^2}{\left(1 - \frac{M(l)}{N}\right)^2} \]
where \( M(l) \) is some kind of degrees of freedom figure. One must take account of both the fitting of the coefficients \( \beta \) in this and the fact that knots (values \( x_{ij} \)) have been chosen. The HTF recommendation is to use
\[ M(l) = 2l + (2 \text{ or } 3) \] (the number of different knots chosen)
(where presumably the knot count refers to different \( x_{ij} \) appearing in at least one \( g_{m1}(x) \) or \( g_{m2}(x) \)).
9 Boosting (Ch10 of HTF)

The basic idea here is to take even a fairly simple (and poorly performing form of) predictor and by sequentially modifying/perturbing it and re-weighting (or modifying) the training data set, to creep toward an effective predictor.

9.1 The AdaBoost Algorithm

Consider a 2-class 0-1 loss classification problem. We’ll suppose that output $Y^*$ takes values in $\mathcal{G} = \{-1, 1\}$ (this coding of the states will be more convenient than \{0, 1\} or \{1, 2\} codings for our present purposes). The AdaBoost.M1 algorithm is built on some base classifier $f$ (that could be, for example, the simplest possible tree classifier, possessing just 2 final nodes). It proceeds as follows.

1. Initialize weights on the training data $(x_i, y_i)$ at

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

2. Fit a $\mathcal{G}$-valued predictor/classifier $f_1$ to the training data to optimize

$$\sum_{i=1}^{N} I[y_i^* \neq f(x_i)]$$

let

$$\overline{err}_1 = \frac{1}{N} \sum_{i=1}^{N} I[y_i^* \neq f_1(x_i)]$$

and define

$$\alpha_1 = \ln \left( \frac{1 - \overline{err}_1}{\overline{err}_1} \right)$$

3. Set new weights on the training data

$$w_{i2} = \frac{1}{N} \exp (\alpha_1 I[y_i^* \neq f_1(x_i)]) \text{ for } i = 1, 2, \ldots, N$$

(This up-weights mis-classified observations by a factor of $(1 - \overline{err}_1) / \overline{err}_1$.)

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

   (a) Fit a $\mathcal{G}$-valued predictor/classifier $f_m$ to the training data to optimize

   $$\sum_{i=1}^{N} w_{im} I[y_i^* \neq f(x_i)]$$

   (b) Let

   $$\overline{err}_m = \frac{\sum_{i=1}^{N} w_{im} I[y_i^* \neq f_m(x_i)]}{\sum_{i=1}^{N} w_{im}}$$
(c) Set
\[ \alpha_m = \ln \left( \frac{1 - \err_m}{\err_m} \right) \]

(d) Update weights as
\[ w_{i(m+1)} = w_{im} \exp (\alpha_m I[y_i \neq f_m(x_i)]) \quad \text{for } i = 1, 2, \ldots, N \]

5. Output a resulting classifier that is based on "weighted voting" by the classifiers \( f_m \) as
\[ f(x) = \sign \left( \sum_{m=1}^{M} \alpha_m f_m(x) \right) = 1 - 2I \left[ \sum_{m=1}^{M} \alpha_m f_m(x) < 0 \right] \]

(Classifiers with small \( \err_m \) get big positive weights in the "voting.")

This apparently turns out to be a very effective way of developing a classifier, and has been called the best existing "off-the-shelf" methodology for the problem.

9.2 Why AdaBoost Might Be Expected to Work

HTF offer some motivation for the effectiveness of AdaBoost. This is related to the optimization of an expected "exponential loss."

9.2.1 Expected "Exponential Loss" for a Function of \( X \)

For \( g \) an arbitrary function of \( X \), consider the quantity
\[ \EE \exp (-Y^* g(X)) = \EE[\exp (-Y^* g(X)) | X] \quad (64) \]

If \( g(X) \) were typically very large positively when \( Y^* = 1 \) and very large negatively when \( Y^* = -1 \), this quantity (64) could be expected to be small. So if one were in possession of such a function of \( X \), one might expect to produce a sensible classifier as
\[ f(x) = \sign(g(x)) = 1 - 2I[g(x) < 0] \quad (65) \]

A bit differently put, a \( g \) constructed to make (64) small should also produce a good classifier as (65).

It’s a bit of a digression, but HTF point out that it’s actually fairly easy to see what \( g(X) \) optimizes (64). One needs only identify for each \( x \) what value \( a \) optimizes
\[ \EE[\exp (-aY^*) | X = x] \]

That is,
\[ \EE[\exp (-aY^*) | X = x] = \exp (-a) P[Y^* = 1 | X = x] + \exp (a) P[Y^* = -1 | X = x] \]

and an optimal \( a \) is easily seen to be half the log odds ratio, i.e. the \( g \) optimizing (64) is
\[ g(x) = \frac{1}{2} \ln \left( \frac{P[Y^* = 1 | X = x]}{P[Y^* = -1 | X = x]} \right) \]
9.2.2 Sequential Search for a $g(X)$ With Small Expected Exponential Loss and the AdaBoost Algorithm

Consider "base classifiers" $h_l(x, \gamma_l)$ (taking values in $\mathcal{G} = \{-1, 1\}$) with parameters $\gamma_l$ and functions built from them

$$g_m(x) = \sum_{l=1}^{m} \beta_l h_l(x, \gamma_l)$$

We might think of successive ones of these as perturbations of the previous ones (by addition of a term $\beta_m h_m(x, \gamma_m)$ to $g_{m-1}(x)$). We might further think about how such functions might be successively defined in an effort to make them effective as producing small values of (64). (Of course, since we typically do not have available a complete probability model for $(X, Y)$ we will only be able to seek to optimize an empirical version of (64).) As it turns out, taking these points of view produces a multiple of the AdaBoost voting function $\sum_{m=1}^{M} \alpha_m f_m(x)$ as $g_M(x)$. (It then follows that AdaBoost might well produce effective classifiers by the logic that $g$ producing small (64) ought to translate to a good classifier via (65).)

So consider an empirical loss that amounts to an empirical version of $N$ times (64) for $g_m$

$$E_m = \sum_{i=1}^{N} \exp (-y_i^* g_m(x_i))$$

This is

$$E_m = \sum_{i=1}^{N} \exp (-y_i^* g_{m-1}(x_i) - y_i^* \beta_m h_m(x_i, \gamma_m))$$

$$= \sum_{i=1}^{N} \exp (-y_i^* g_{m-1}(x_i)) \exp (-y_i^* \beta_m h_m(x, \gamma_m))$$

Let

$$v_{im} = \exp (-y_i^* g_{m-1}(x_i))$$

and consider optimal choice of $\gamma_m$ and $\beta_m > 0$ (for purposes of making $g_m$ the best possible perturbation of $g_{m-1}$). (We may assume that $\beta > 0$ provided the set of classifiers defined by $h_l(x, \gamma_l)$ as $\gamma_l$ varies includes both $h$ and $-h$ for any version of the base classifier, $h$.) Write

$$E_m = \sum_{i \text{ with } h_m(x_i, \gamma_m) = y_i^*} v_{im} \exp (-\beta_m) + \sum_{i \text{ with } h_m(x_i, \gamma_m) \neq y_i^*} v_{im} \exp (\beta_m)$$

$$= (\exp(\beta_m) - \exp(-\beta_m)) \sum_{i=1}^{N} v_{im} I[h_m(x_i, \gamma_m) \neq y_i^*] + \exp(-\beta_m) \sum_{i=1}^{N} v_{im}$$

So independent of the value of $\beta_m > 0$, $\gamma_m$ needs to be chosen to optimize the $v_{im}$-weighted error rate of $h_m(x, \gamma_m)$. Call the optimized (using weights
$v_{im}$) version of $h_m (x, \gamma_m)$ by the name $h_m (x)$, and notice that the prescription for choice of this base classifier is exactly the AdaBoost prescription of how to choose $f_m$ (based on weights $w_{im}$).

So now consider minimization of

$$\sum_{i=1}^{N} \exp (-y_i^* \beta h_m (x_i))$$

over choices of $\beta$. This is

$$\exp (-\beta) \left( \sum_{i \text{ with } h_m (x_i, \gamma_m) = y_i^*} v_{im} + \sum_{i \text{ with } h_m (x_i, \gamma_m) \neq y_i^*} v_{im} \exp (2\beta_m) \right)$$

$$= \exp (-\beta) \left( \sum_{i=1}^{N} v_{im} + \sum_{i=1}^{N} v_{im} (\exp (2\beta_m) - 1) I [h_m (x_i) \neq y_i^*] \right)$$

and minimization of it is equivalent to minimization of

$$\exp (-\beta) \left( 1 + (\exp (2\beta_m) - 1) \frac{\sum_{i=1}^{N} v_{im} I [h_m (x_i) \neq y_i^*]}{\sum_{i=1}^{N} v_{im}} \right)$$

Let

$$\frac{\text{err}}{\sum_{i=1}^{N} v_{im} I [h_m (x_i) \neq y_i^*]} = \frac{\sum_{i=1}^{N} v_{im} I [h_m (x_i) \neq y_i^*]}{\sum_{i=1}^{N} v_{im}}$$

and see that choice of $\beta$ requires minimization of

$$\exp (-\beta) \left( 1 + (\exp (2\beta_m) - 1) \frac{\sum_{i=1}^{N} v_{im} I [h_m (x_i) \neq y_i^*]}{\sum_{i=1}^{N} v_{im}} \right)$$

A bit of calculus shows that the optimizing $\beta$ is

$$\beta_m = \frac{1}{2} \ln \left( 1 - \frac{\text{err}}{\sum_{i=1}^{N} v_{im} I [h_m (x_i) \neq y_i^*]} \right)$$

(66)

Notice that the prescription (66) calls for a coefficient that is derived as exactly half $\alpha_m$, the coefficient of $f_m$ in the AdaBoost voting function (starting from weights $w_{im}$) prescribed in 4b. of the AdaBoost algorithm. (And note that as regards sign, the $1/2$ is completely irrelevant.)

So to finish the argument that successive choosing of $g_m$'s amounts to AdaBoost creation of a voting function, it remains to consider how the weights $v_{im}$ get updated. For moving from $g_m$ to $g_{m+1}$ one uses weights

$$v_{i(m+1)} = \exp (-y_i^* g_m (x_i))$$

$$= \exp (-y_i^* (g_{m-1} (x_i) + \beta_m h_m (x_i)))$$

$$= v_{im} \exp (-y_i^* \beta_m h_m (x_i))$$

$$= v_{im} \exp (\beta_m (2I [h_m (x_i) \neq y_i^*] - 1))$$

$$= v_{im} \exp (2\beta_m I [h_m (x_i) \neq y_i^*]) \exp (\beta_m)$$

72
Since $\exp(\beta_m)$ is constant across $i$, it is irrelevant to weighting, and since the prescription for $\beta_m$ produces half what AdaBoost prescribes in 4b. for $\alpha_m$ the weights used in the choice of $\beta_{m+1}$ and $h_{m+1}(x, \gamma_{m+1})$ are derived exactly according to the AdaBoost algorithm for updating weights.

So, at the end of the day, $g_m$’s are built from base classifiers to creep towards a function optimizing an empirical version of $E\exp(-Y^*g(X))$. Since it is easy to see that $g_1$ corresponds to the first AdaBoost step, $g_m$ is further $1/2$ of the AdaBoost voting function, they generate the same classifier as the AdaBoost algorithm.

### 9.3 Other Forms of Boosting

One might, in general, think about strategies for sequentially modifying/perturbing a predictor (perhaps based on a modified version of a data set) as a way of creeping towards a good predictor. Here is a general "gradient boosting algorithm" (taken from Izenman) aimed in this direction for a general loss function and general class of potential updates for a predictor. (I don’t see that HTF ever get to the point of actually laying out this logic, though they say lots of related things!)

1. Start with $f_0(x) = \arg\min_{\tilde{y}} \sum_{i=1}^N L(y_i, \tilde{y})$.

2. For $m = 1, 2, \ldots, M$

   (a) let $\tilde{y}_{im} = - \frac{\partial}{\partial \tilde{y}} L(y_i, \tilde{y}) \bigg|_{\tilde{y}=f_{m-1}(x_i)}$

   (b) fit, for some class of functions $h_m(\cdot, \cdot)$, parameters $\beta_m$ and $\gamma_m$ as

   $$\min_{(\beta, \gamma)} \sum_{i=1}^N (\tilde{y}_{im} - \beta h_m(x_i, \gamma))^2$$

   (c) let

   $$\rho_m = \arg\min_{\rho} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + \rho h_m(x_i, \gamma_m))$$

   (d) set

   $$f_m(x) = f_{m-1}(x) + \rho_m h_m(x, \gamma_m)$$

3. and finally, set $f(x) = f_M(x)$.

In this algorithm, the set of $N$ values $\beta_m h_m(x_i, \gamma_m)$ in 2b. functions as an approximate negative gradient of the total loss at the set of values $f_{m-1}(x_i)$ and the least squares criterion at 2b. might be replaced by others (like least absolute deviations). The "minimizations" need to be taken with a grain of salt, as they are typically only the "approximate" minimizations (like those
associated with greedy algorithms used to fit binary trees, e.g.). The step 2d.
is some kind of line search in a direction of steepest descent for the total loss.

Despite the fact that the HTF Section 10.9 is titled "Boosting Trees," there is
nothing here that requires that the functions $\beta h_m (x, \gamma)$ come from classification
or regression trees. When the problem is a regression problem with squared
error loss some substantial simplifications of this occur.

9.3.1 SEL

Suppose for now that $L (y, \tilde{y}) = \frac{1}{2} (y - \tilde{y})^2$. Then, in the gradient boosting
algorithm above

1. $f_0 (x) = \tilde{y},$

2. $\tilde{y}_{im} = - \frac{\partial}{\partial \tilde{y}} \left( \frac{1}{2} (y_i - \tilde{y})^2 \right) \Big|_{\tilde{y} = f_{m-1} (x_i)} = y_i - f_{m-1} (x_i) \text{ so that step 2b.}$
of the algorithm says "fit, using SEL, a model for the residuals from the
previous iteration," and

3. the optimization of $\rho$ in step 3c of the algorithm produces $\rho_m = \beta_m$ so
that step 3d of the algorithm sets

$$f_m (x) = f_{m-1} (x) + \beta_m h_m (x, \gamma_m)$$

and ultimately one is left with the prescription "fit, using, SEL a model for the
residuals from the previous iteration and add it to the previous predictor to
produce the $m$th one," a very intuitively appealing algorithm.

Notice that still, there is nothing here that really requires that the $\beta h_m (x, \gamma)$
come from regression trees. This notion of gradient boosting could be applied
to any kind of base regression predictor.

9.3.2 AEL (and Binary Regression Trees)

Suppose now that $L (y, \tilde{y}) = |y - \tilde{y}|$. Then, in the gradient boosting algorithm

1. $f_0 (x) = \text{median } \{y_i\}, \text{ and}$

2. $\tilde{y}_{im} = - \frac{\partial}{\partial \tilde{y}} (|y_i - \tilde{y}|) \Big|_{\tilde{y} = f_{m-1} (x_i)} = \text{sign}(y_i - f_{m-1} (x_i)) \text{ so that step 2b.}$
of the algorithm says "fit, a model for $\pm 1$'s coding the signs of the residuals
from the previous iteration."

In the event that the base predictors being fit are regression trees, the
$\beta_m h_m (x, \gamma_m)$ fit at step 2b. will be values between $-1$ and $1$ (constant on
rectangles) if the fitting is done using least squares (exactly as indicated in 2b.).
If the alternative of least absolute error were used in place of least squares in
2b., one would have the median of a set of $-1$'s and $1$'s, and thus values either
−1 or 1 constant on rectangles. In this latter case, the optimization for \( \rho_m \) is of the form

\[
\rho_m = \arg \min_{\rho} \sum_{i=1}^{N} \left| y - f_{m-1}(x_i) - \frac{\rho}{\beta_m} (\pm 1)_i \right|
\]

and while there is no obvious closed form for the optimizer, it is at least the case that one is minimizing a piecewise linear continuous function of \( \rho \).

### 9.3.3 \( K \)-Class Classification

HTF allude to a procedure that uses gradient boosting in fitting a \((K\)-dimensional\) predictor

\[
f = (f_1, \ldots, f_K)
\]

under the constraint that \( \sum_{k=1}^{K} f_k = 0 \), sets corresponding class probabilities (for \( k \in \mathcal{G} = \{1, 2, \ldots, K\} \)) to

\[
p_k(x) = \frac{\exp(f_k(x))}{\sum_{k=1}^{K} \exp(f_k(x))}
\]

and has the implied classifier

\[
f(x) = \arg \max_{k \in \mathcal{G}} p_k(x)
\]

An appropriate (and convenient) loss is

\[
L(y^*, p) = - \sum_{k=1}^{K} I[y^* = k] \ln(p_k)
\]

\[
= - \sum_{k=1}^{K} I[y^* = k] f_k + \ln \left( \sum_{k=1}^{K} \exp(f_k) \right)
\]

This leads to

\[
-\frac{\partial}{\partial f_k} L(y^*, p) = I[y^* = k] - \frac{\exp(f_k)}{\sum_{l=1}^{K} \exp(f_l)}
\]

\[
= I[y^* = k] - p_k
\]

and thus

\[
\tilde{y}_{ikm} = I[y^*_i = k] - p_{k(m-1)}(x_i)
\]

Algorithm 10.4 on page 387 then provides a version of the gradient boosting algorithm for this loss and \( K \)-class classification.
9.4 Issues (More or Less Related to Boosting and Use of Trees as Base Predictors)

The balance of Ch 10 of HTF concerns a number of issues that might come up in the use of boosting, particularly if regression or classification trees are the base predictors.

One issue is the question of how large regression or classification trees should be allowed to grow if they are used as the functions $\beta h_m(x, \gamma)$ in a boosting algorithm. The answer seems to be "Not too large, maybe to about 6 or so terminal nodes."

Section 10.12 concerns regularization/control of complexity parameters/avoiding overfit in boosting. Where trees are used as the base predictors, limiting their size (as above) provides some regularization. Further, there is the parameter $M$ in the boosting algorithm that can/should be limited in size (very large values surely producing overfit). Cross-validation here looks typically pretty unworkable because of the amount of computing involved. HTF seem to indicate that (for better or worse?) holding back a part of the training sample and watching performance of a predictor on that single test set as $M$ increases is a commonly used method of choosing $M$.

"Shrinkage" is an idea that has been suggested for regularization in boosting. The basic idea is that instead of setting

$$f_m(x) = f_{m-1}(x) + \rho_m h_m(x, \gamma_m)$$

as in 2d. of the gradient boosting algorithm, one chooses $\nu \in (0,1)$ and sets

$$f_m(x) = f_{m-1}(x) + \nu \rho_m h_m(x, \gamma_m)$$

i.e. one doesn't make the "full correction" to $f_{m-1}$ in producing $f_m$. This will typically require larger $M$ for good final performance of a predictor than the non-shrunken version, but apparently can (in combination with larger $M$) improve performance.

"Subsampling" is some kind of bootstrap/bagging idea. The notion is that at each iteration of the boosting algorithm, instead of choosing an update based on the whole training set, one chooses a fraction $\eta$ of the training set at random, and fits to it (using a new random selection at each iteration). This reduces the computation time per iteration and apparently can also improve performance. HTF show its performance on an example in combination with shrinkage.

Consider the matter of assigning measures of "importance" of input variables for a regression tree predictor. In the spirit of ordinary linear models assessment of the importance of a predictor in terms of some reduction it provides in some error sum of squares, Breiman suggested the following. Suppose that in a regression tree, predictor $X_j$ provides the rectangle splitting criterion for nodes $node_{1j}, \ldots, node_{mj}$ and that before splitting at $node_{ij}$, the relevant rectangle $R_{node_{ij}}$ has associated sum of squares

$$SSE_{node_{ij}} = \sum_{i \text{ with } x_i \in R_{node_{ij}}} (y_i - \overline{y}_{node_{ij}})^2$$
and that after splitting $R_{\text{node}_{ij}}$ on variable $X_j$ to create rectangles $R_{\text{node}_{ij}}$ and $R_{\text{node}_{ij}}$ one has sums of squares associated with those two rectangles

$$SSE_{\text{node}_{ij}} = \sum_{i \text{ with } x_i \in R_{\text{node}_{ij}}} (y_i - \bar{y}_{\text{node}_{ij}})^2$$

and

$$SSE_{\text{node}_{ij}} = \sum_{i \text{ with } x_i \in R_{\text{node}_{ij}}} (y_i - \bar{y}_{\text{node}_{ij}})^2$$

The reduction in error sum of squares provided by the split on $X_j$ at $\text{node}_{ij}$ is thus

$$D(\text{node}_{ij}) = SSE_{\text{node}_{ij}} - (SSE_{\text{node}_{ij}} + SSE_{\text{node}_{ij}})$$

One might then call

$$I_j = \sum_{l=1}^{m(j)} D(\text{node}_{lj})$$

a measure of the importance of $X_j$ in fitting the tree and compare the various $I_j$'s.

Further, if a predictor is a sum of regression trees produced by boosting and $I_{jm}$ measures the importance of $X_j$ in the $m$th tree, then

$$I_{j} = \frac{1}{M} \sum_{m=1}^{M} I_{jm}$$

is a sensible measure of the importance of $X_j$ in the overall predictor. One can then compare the various $I_j$ as a means of comparing the importance of the input variables.

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

$$\bar{f}_{12}(x_1, x_2) = Ef(x_1, x_2, X_3, X_4, \ldots, X_p)$$

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

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

This might be plotted and the plot called a partial dependence plot. HTF's language is that this details the dependence of the predictor on $(x_1, x_2)$ "after accounting for the average effects of the other variables."
Something different is

$$\tilde{f}_{12}(x_1, x_2) = \text{E}[f(X) | X_1 = x_1, X_2 = x_2]$$

(This is, by the way, the function of \((X_1, X_2)\) closest to \(f(X)\) in \(L_2(P)\).) This is obtained by averaging not against the marginal of the excluded variables, but against the conditional distribution of the excluded variables given \(X_1 = x_1, X_2 = x_2\). No workable empirical version of \(\tilde{f}_{12}(x_1, x_2)\) can typically be defined. And it should be clear that this is not the same as \(\tilde{f}_{12}(x_1, x_2)\). HTF say this in some sense describes the effects of \((x_1, x_2)\) on the prediction "ignoring the impact of the other variables."

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

$$f(x) = h_1(x_1, x_2) + h_2(x_3, \ldots, x_p)$$

then

$$f_{12}(x_1, x_2) = h_1(x_1, x_2) + \text{E}[h_2(X_3, \ldots, X_p) | X_1 = x_1, X_2 = x_2]$$

while

$$\tilde{f}_{12}(x_1, x_2) = h_1(x_1, x_2) + \text{E}[h_2(X_3, \ldots, X_p) | X_1 = x_1, X_2 = x_2]$$

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

10 Neural Networks (Ch 11 of HTF)

10.1 Projection Pursuit Regression

For \(w_1, w_2, \ldots, w_M\) unit \(p\)-vectors of parameters, we consider predictors of the form

$$f(X) = \sum_{m=1}^{M} g_m(w_m'X)$$

(67)

This is an additive form in the derived variables \(v_m = w_m'X\). The functions \(g_m\) and the directions \(w_m\) are to be fit from the training data. The \(M = 1\) case of this form is the "single index model" of econometrics.

How does one fit a predictor of this form (67)? Consider first the \(M = 1\) case. Given \(w\), we simply have pairs \((v_i, y_i)\) for \(v_i = w'x_i\) and a 1-dimensional smoothing method can be used to estimate \(g\). On the other hand, given \(g\), we might seek to optimize \(w\) via an iterative search. A Gauss-Newton algorithm can be based on the first order Taylor approximation

$$g(w'x_i) \approx g(w'_{\text{old}}x_i) + g'(w'_{\text{old}}x_i)(w' - w'_{\text{old}})x_i$$
so that
\[ \sum_{i=1}^{N} (y_i - g(w' x_i))^2 \approx \sum_{i=1}^{N} (g'(w'_\text{old} x_i))^2 \left( \frac{w'_\text{old} x_i + y_i - g(w'_\text{old} x_i)}{g'(w'_\text{old} x_i)} - w' x_i \right)^2 \]

We may then update \( w_{\text{old}} \) to \( w \) using the closed form for weighted (by \( (g(w' \text{old} x_i))^2 \)) no-intercept regression of
\[ \left( w'_\text{old} x_i + \frac{y_i - g(w'_\text{old} x_i)}{g'(w'_\text{old} x_i)} \right) \]
on \( x_i \). (Presumably one must normalize the updated \( w \) in order to preserve unit length property of the \( w \) in order to maintain a stable scaling in the fitting.) The \( g \) and \( w \) steps are iterated until convergence.

When \( M > 1 \), terms \( g_m(w'_m X) \) are added to a sum of such in a forward stage-wise fashion. HTF provide some discussion of details like readjusting previous \( g \)’s (and perhaps \( w \)’s) upon adding \( g_m(w'_m X) \) to a fit, and the choice of \( M \).

### 10.2 Neural Networks

A multi-layer feed-forward neural network is a nonlinear map of \( X \in \mathbb{R}^p \) to one or more real-valued \( Y \)'s through the use of functions of linear combinations of functions of linear combinations of ... of functions of linear combinations of coordinates of \( X \). Figure 1 is a network diagram representation of a single hidden layer feed-forward neural net with 3 inputs 2 hidden nodes and 2 outputs. This diagram stands for a function of \( X \) defined by setting
\[
Z_1 = \sigma(\alpha_{01} \cdot 1 + \alpha_{11} X_1 + \alpha_{21} X_2 + \alpha_{31} X_3) \\
Z_2 = \sigma(\alpha_{02} \cdot 1 + \alpha_{12} X_1 + \alpha_{22} X_2 + \alpha_{32} X_3)
\]
and then
\[
Y_1 = g_1(\beta_{01} \cdot 1 + \beta_{11} Z_1 + \beta_{21} Z_2) \\
Y_2 = g_1(\beta_{02} \cdot 1 + \beta_{12} Z_1 + \beta_{22} Z_2)
\]
Of course, much more complicated networks are possible, particularly ones with multiple hidden layers and many nodes on all layers.

The common choices of functional form \( \sigma \) at hidden nodes are ones with sigmoidal shape like
\[
\sigma(u) = \frac{1}{1 + \exp(-u)}
\]
or
\[
\sigma(u) = \tanh(u) = \frac{\exp(u) - \exp(-u)}{\exp(u) + \exp(-u)}
\]
(statisticians seem to favor the former, probably because of familiarity with logistic forms, while computer scientists seem to favor the latter). Such functions
are really quite linear near \( u = 0 \), so that for small \( \alpha \)'s the functions of \( \mathbf{X} \) entering the \( g \)'s in a single hidden layer network are nearly linear. For large \( \alpha \)'s the functions are nearly step functions in \( \alpha \mathbf{X} \). In light of the latter, it is not surprising that there are universal approximation theorems that guarantee that any continuous function on a compact subset of \( \mathbb{R}^p \) can be approximated to any degree of fidelity with a single layer feed-forward neural net with enough nodes in the hidden layer. This is both a blessing and a curse. It promises that these forms are quite flexible. It also promises that there must be both over-fitting and identifiability issues inherent in their use (the latter in addition to the identifiability issues already inherent in the symmetric nature of the functional forms assumed for the predictors). In regression contexts, identity functions are common for the functions \( g \), while in classification problems, in order to make estimates of class probabilities, the functions \( g \) often exponentiate one \( Z \) and divide by a sum of such terms for all \( Z \)'s.

There are various possibilities for regularization of the ill-posed fitting problem for neural nets, ranging from the fairly formal and rational to the very informal and ad hoc. Possibly the most common is to simply use an iterative fitting algorithm and "stop it before it converges."

### 10.2.1 The Back-propagation Algorithm

The most common fitting algorithm is something called the back-propagation algorithm or the delta rule. In the case where one is doing SEL fitting for a single layer feed-forward neural net with \( M \) hidden nodes and \( K \) output nodes,
the logic goes as follows. There are \( M(p + 1) \) parameters \( \alpha \) to be fit. In addition to the notation \( \alpha_{0m} \) use the notation
\[
\alpha_m = (\alpha_{1m}, \alpha_{2m}, \ldots, \alpha_{pm})' 
\]
There are \( K(M + 1) \) parameters \( \beta \) to be fit. In addition to the notation \( \beta_{0k} \), use the notation
\[
\beta_k = (\beta_{1k}, \beta_{2k}, \ldots, \beta_{MK})'
\]
Let \( \theta \) stand for the whole set of parameters and consider the SEL fitting criterion
\[
R(\theta) = \frac{1}{2} \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2 \tag{68}
\]
Let \( z_{0i} = 1 \) and for \( m = 1, 2, \ldots, M \) define
\[
z_{mi} = \sigma(\alpha_{0m} + \alpha'_m x_i) 
\]
and
\[
z_i = (z_{1i}, z_{2i}, \ldots, z_{Mi})'
\]
and write
\[
R_i = \frac{1}{2} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2 
\]
(so that \( R(\theta) = \sum_{i=1}^{N} R_i \)).
Partial derivatives with respect to the parameters are (by the chain rule)
\[
\frac{\partial R_i}{\partial \beta_{mk}} = - (y_{ik} - f_k(x_i)) g'_k (\beta_{0k} + \beta'_k z_i) z_{mi} 
\]
(for \( m = 0, 1, 2, \ldots, M \) and \( k = 1, 2, \ldots, K \)) and we will use the abbreviation
\[
\delta_{ki} = - (y_{ik} - f_k(x_i)) g'_k (\beta_{0k} + \beta'_k z_i) 
\]
so that
\[
\frac{\partial R_i}{\partial \beta_{mk}} = \delta_{ki} z_{mi} \tag{69}
\]
Similarly, letting \( x_{i0} = 1 \)
\[
\frac{\partial R_i}{\partial \alpha_{jm}} = - \sum_{k=1}^{K} (y_{ik} - f_k(x_i)) g'_k (\beta_{0k} + \beta'_k z_i) \beta_{mk} \sigma' (\alpha_{0m} + \alpha'_m x_i) x_{il} 
\]
\[
= \sum_{k=1}^{K} \delta_{ki} \beta_{mk} \sigma' (\alpha_{0m} + \alpha'_m x_i) x_{il} 
\]
(for \( l = 0, 1, 2, \ldots, p \) and \( m = 1, 2, \ldots, M \)) and we will use the abbreviations

\[
\begin{align*}
    s_{mi} &= -\sum_{k=1}^{K} (y_{ik} - f_k(x_i)) g_k(\beta_{0k} + \beta_k z_i) \beta_{mk} \sigma'(\alpha_{0m} + \alpha'_m x_i) \\
    &= \sum_{k=1}^{K} \beta_{mk} \sigma'(\alpha_{0m} + \alpha'_m x_i) \delta_{ki}
\end{align*}
\]

(equations (71) are called the "back-propagation" equations) so that

\[
\frac{\partial R_i}{\partial \alpha_{lm}} = s_{mi} x_{il}
\]

An iterative search to make \( R(\theta) \) small may then proceed by setting

\[
\beta_{mk}^{(r+1)} = \beta_{mk}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{mk}} |_{\beta_{0k}^{(r)}, \beta_{k}^{(r)}} = \beta_{mk}^{(r)} - \gamma_r \sum_{i=1}^{N} \delta_{ki} z_{mi}^{(r)} \quad \forall m, k
\]

and

\[
\alpha_{lm}^{(r+1)} = \alpha_{lm}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{lm}} |_{\alpha_{0m}^{(r)}, \alpha_{m}^{(r)}, \beta_{0k}^{(r)}, \beta_{k}^{(r)}}
\]

\[
= \alpha_{lm}^{(r)} - \gamma_r \sum_{i=1}^{N} s_{mi}^{(r)} x_{il}
\]

for some "learning rate" \( \gamma_r \).

So operationally

1. in what is called the "forward pass" through the network, using \( r \)th iterates of \( \alpha \)'s and \( \beta \)'s, one computes \( r \)th iterates of fitted values \( f_k^{(r)}(x_i) \),

2. in what is called the "backward pass" through the network, the \( (r) \)th iterates \( \delta_{ki}^{(r)} \) are computed using the \( r \)th iterates of the \( z \)'s, \( \beta \)'s and the \( f_k^{(r)}(x_i) \) in equations (69) and then the \( (r) \)th iterates \( s_{ml}^{(r)} \) are computed using the \( (r) \)th \( \delta \)'s, \( \alpha \)'s, and \( \beta \)'s in equations (71),

3. the \( \delta_{ki}^{(r)} \) and \( s_{ml}^{(r)} \) then provide partials using (70) and (72), and

4. updates for the parameters \( \alpha \) and \( \beta \) come from (73) and (74).

### 10.2.2 Formal Regularization of Fitting

Suppose that the various coordinates of the input vectors in the training set have been standardized and one wants to regularize the fitting of a neural net. One possible way of proceeding is to define a penalty function like

\[
J(\theta) = \frac{1}{2} \left( \sum_{i=0}^{P} \sum_{m=1}^{M} \alpha_{lm}^2 + \sum_{m=0}^{M} \sum_{k=1}^{K} \beta_{mk}^2 \right)
\]
(it is not absolutely clear whether one really wants to include \( l = 0 \) and \( m = 0 \) in the sums (75)) and seek not to partially optimize \( R(\theta) \) in (68), but rather to fully optimize

\[
R(\theta) + \lambda J(\theta)
\]

for a \( \lambda > 0 \). By simply adding \( -\gamma_l \lambda _m\beta_m \) on the right side of (73) and \( -\gamma_l \lambda _m\alpha_m \) on the right side of (74), one arrives at an appropriate gradient descent algorithm for trying to optimize the penalized error sum of squares (76). (Potentially, an appropriate value for \( \lambda \) might be chosen based on cross-validation.)

Something that may at first seem quite different would be to take a Bayesian point of view. For example, with a model

\[
y_{ik} = f_k(x_i; \theta) + \epsilon_{ik}
\]

for the \( \epsilon_{ik} \) iid \( N(0, \sigma^2) \), a likelihood is simply

\[
l(\theta, \sigma^2) = \frac{1}{NK} \prod_{i=1}^{N} \prod_{k=1}^{K} h(y_{ik}|f_k(x_i; \theta), \sigma^2)
\]

for \( h(\cdot|\mu, \sigma^2) \) the normal pdf. If then \( g(\theta, \sigma^2) \) specifies a prior distribution for \( \theta \) and \( \sigma^2 \), a posterior for \( (\theta, \sigma^2) \) has density proportional to

\[
l(\theta, \sigma^2) g(\theta, \sigma^2)
\]

For example, one might well assume that a priori the \( \alpha \)'s and \( \beta \)'s are iid \( N(0, \eta^2) \) (where small \( \eta^2 \) will provide regularization and it is again unclear whether one wants to include the \( \alpha_0m \) and \( \beta_0k \) in such an assumption or to instead provide more diffuse priors for them, like improper "Uniform\((-\infty, \infty)\)" or at least large variance normal ones). And a standard (improper) prior for \( \sigma^2 \) would be to take \( \log \sigma \) to be Uniform\((-\infty, \infty)\). In any case, whether improper or proper, let’s abuse notation and write \( g(\sigma^2) \) for the prior density for \( \sigma^2 \). Then with independent mean 0 variance \( \eta^2 \) priors for all the weights (except possibly the \( \alpha_0m \) and \( \beta_0k \) that might be given Uniform\((-\infty, \infty) \) priors) one has

\[
\ln l(\theta, \sigma^2) g(\theta, \sigma^2) \\
= -NK \ln(\sigma) - \frac{1}{\sigma^2} R(\theta) - \frac{1}{\eta^2} J(\theta) + \ln g(\sigma^2)
\]

(Flat improper priors for the \( \alpha_0m \) and \( \beta_0k \) correspond to the absence of terms for them in the sums for \( J(\theta) \) in (75)). This recalls (76) and suggests that appropriate \( \lambda \) for regularization can be thought of as a variance ratio of "observation variance" and prior variance for the weights.

It’s fairly clear how to define Metropolis-Hastings-within-Gibbs algorithms for sampling from \( l(\theta, \sigma^2) g(\theta, \sigma^2) \). But it seems that typically the high
dimensionality of the parameter space combined with the symmetry-derived multi-modality of the posterior will typically prevent one from running an MCMC algorithm long enough to fully detail the posterior. It also seems unlikely however, that detailing the posterior is really necessary or even desirable. Rather, one might simply run the MCMC algorithm, monitoring the values of \( l(\theta, \sigma^2) g(\theta, \sigma^2) \) corresponding to the successively randomly generated MCMC iterates. An MCMC algorithm will spend much of its time where the corresponding posterior density is large and we can expect that a long MCMC run will identify a nearly modal value for the posterior. Rather than averaging neural nets according to the posterior, one might instead use as a predictor a neural net corresponding to a parameter vector (at least locally) maximizing the posterior. Notice that one might even take the parameter vector in an MCMC run with the largest \( l(\theta, \sigma^2) g(\theta, \sigma^2) \) value and for a grid of \( \sigma^2 \) values around the empirical maximizer use the back-propagation algorithm modified to fully optimize

\[
R(\theta) + \frac{\sigma^2}{\eta^2} J(\theta)
\]

over choices of \( \theta \). This, in turn, could be used with (77) to perhaps improve somewhat the result of the MCMC "search."

11 Support Vector Machines (Ch 12 of HTF)

Consider a 2-class classification problem. As for the AdaBoost derivation, we’ll suppose that output \( Y^* \) takes values in \( \mathcal{G} = \{-1, 1\} \). Our present concern is in a further development of linear classification methodology beyond that provided in Section 3.

For \( \beta \in \mathbb{R}^p \) and \( \beta_0 \in \mathbb{R} \) we’ll consider the form

\[
g(x) = x'\beta + \beta_0
\]

and a predictor/classifier

\[
f(x) = \text{sign}(g(x))
\]

We will approach the problem of choosing \( \beta \) and \( \beta_0 \) to in some sense provide a maximal cushion around a hyperplane separating between \( x_i \) with corresponding \( y_i^* = -1 \) and \( x_i \) with corresponding \( y_i^* = 1 \).

11.1 The Linearly Separable Case

In the case that there is a classifier of form (79) with 0 training error rate, we consider the optimization problem

\[
\begin{array}{ll}
\text{maximize} & M \\
\text{subject to} & y_i^* (x_i'\beta + \beta_0) \geq M \ \forall i
\end{array}
\]

(80)
This can be thought of in terms of choosing a unit vector $\mathbf{u}$ (or direction) in $\mathbb{R}^p$ so that upon projecting the training input vectors $\mathbf{x}_i$ onto the subspace of multiples of $\mathbf{u}$ there is maximum separation between convex hull of projections of the $\mathbf{x}_i$ with $y_i^* = -1$ and the convex hull of projections of $\mathbf{x}_i$ with corresponding $y_i^* = 1$. (The sign on $\mathbf{u}$ is chosen to give the latter larger $\mathbf{x}_i' \mathbf{u}$ than the former.) Notice that if $\mathbf{u}$ and $\beta_0$ solve this minimization problem

$$M = \frac{1}{2} \left( \min_{\mathbf{x}_i \text{ with } y_i^* = 1} \mathbf{x}_i' \mathbf{u} - \max_{\mathbf{x}_i \text{ with } y_i^* = -1} \mathbf{x}_i' \mathbf{u} \right)$$

and

$$\beta_0 = -\frac{1}{2} \left( \min_{\mathbf{x}_i \text{ with } y_i^* = 1} \mathbf{x}_i' \mathbf{u} + \max_{\mathbf{x}_i \text{ with } y_i^* = -1} \mathbf{x}_i' \mathbf{u} \right)$$

For purposes of applying standard optimization theory and software, it is useful to reformulate the basic problem (80) several ways. First, note that (80) may be rewritten as

$$\max_{\mathbf{u}} M \text{ subject to } y_i^* \left( \mathbf{x}_i' \left( \frac{\mathbf{u}}{M} \right) + \frac{\beta_0}{M} \right) \geq 1 \ \forall i \quad (81)$$

Then if we let

$$\beta = \frac{\mathbf{u}}{M}$$

it’s the case that

$$\|\beta\| = \frac{1}{M} \quad \text{or} \quad M = \frac{1}{\|\beta\|}$$

so that (81) can be rewritten

$$\min_{\beta \in \mathbb{R}^p} \frac{1}{2} \|\beta\|^2 \text{ subject to } y_i^* (\mathbf{x}_i' \beta + \beta_0) \geq 1 \ \forall i \quad (82)$$

and $\beta_0 \in \mathbb{R}$

This formulation (82) is that of a convex (quadratic criterion, linear inequality constraints) optimization problem for which there exists standard theory and algorithms.

The so-called primal functional corresponding to (82) is (for $\alpha \in \mathbb{R}^N$)

$$F_P(\beta, \beta_0, \alpha) = \frac{1}{2} \|\beta\|^2 - \sum_{i=1}^N \alpha_i (y_i^* (\mathbf{x}_i' \beta + \beta_0) - 1) \quad \text{for} \quad \alpha \geq 0$$

85
To solve (82), one may for each \( \alpha \geq 0 \) choose \((\beta(\alpha), \beta_0(\alpha))\) to minimize \( F_P(\cdot, \cdot, \alpha) \) and then choose \( \alpha \geq 0 \) to maximize \( F_P(\beta(\alpha), \beta_0(\alpha), \alpha) \). The Karush-Kuhn-Tucker conditions are necessary and sufficient for solution of a constrained optimization problem. In the present context they are the gradient conditions

\[
\frac{\partial F_P(\beta, \beta_0, \alpha)}{\partial \beta_0} = -\sum_{i=1}^{N} \alpha_i y_i^* = 0 \tag{83}
\]

and

\[
\frac{\partial F_P(\beta, \beta_0, \alpha)}{\partial \beta} = \beta - \sum_{i=1}^{N} \alpha_i y_i^* x_i = 0 \tag{84}
\]

the feasibility conditions

\[
y_i^* (x_i^* \beta + \beta_0) - 1 \geq 0 \quad \forall i \tag{85}
\]

the non-negativity conditions

\[
\alpha \geq 0 \tag{86}
\]

and the orthogonality conditions

\[
\alpha_i (y_i^* (x_i^* \beta + \beta_0) - 1) = 0 \quad \forall i \tag{87}
\]

Now (83) and (84) are

\[
\sum_{i=1}^{N} \alpha_i y_i^* = 0 \quad \text{and} \quad \beta = \sum_{i=1}^{N} \alpha_i y_i^* x_i \equiv \beta(\alpha) \tag{88}
\]

and plugging these into \( F_P(\beta, \beta_0, \alpha) \) gives a function of \( \alpha \) only

\[
F_D(\alpha) = \frac{1}{2} ||\beta(\alpha)||^2 - \sum_{i=1}^{N} \alpha_i (y_i^* x_i^* \beta(\alpha) - 1)
\]

\[
= \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i^* y_j^* x_i x_j - \sum_i \sum_j \alpha_i \alpha_j y_i^* y_j^* x_i x_j + \sum_i \alpha_i
\]

\[
= \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i^* y_j^* x_i x_j
\]

\[
= \mathbf{1}' \alpha - \frac{1}{2} \alpha' H \alpha \tag{89}
\]

for

\[
H_{N \times N} = (y_i^* y_j^* x_i^* x_j) \tag{90}
\]

Then the "dual" problem is

\[
\text{maximize} \quad \mathbf{1}' \alpha - \frac{1}{2} \alpha' H \alpha \quad \text{subject to} \quad \alpha \geq 0 \quad \text{and} \quad \alpha' y^* = 0 \tag{90}
\]
and apparently this problem is easily solved.  

Now condition (87) implies that if $\alpha_i^{opt} > 0$

$$y_i^* (x'_i \beta(\alpha^{opt}) + \beta_0(\alpha^{opt})) = 1$$

so that 

1. by (85) the corresponding $x_i$ has minimum $x'_i \beta(\alpha^{opt})$ for training vectors with $y_i^* = 1$ or maximum $x'_i \beta(\alpha^{opt})$ for training vectors with $y_i^* = -1$ (so that $x_i$ is a support vector for the "slab" of thickness $2M$ around a separating hyperplane),

2. $\beta_0(\alpha^{opt})$ may be determined using the corresponding $x_i$ from

$$y_i^* \beta_0(\alpha^{opt}) = 1 - y_i^* x'_i \beta(\alpha^{opt})$$

i.e. $\beta_0(\alpha^{opt}) = y_i^* - x'_i \beta(\alpha^{opt})$

(apparently for reasons of numerical stability it is common practice to average values $y_i^* - x'_i \beta(\alpha^{opt})$ for support vectors in order to evaluate $\beta_0(\alpha^{opt})$) and,

3. 

$$1 = y_i^* \beta_0(\alpha^{opt}) + y_i^* \left( \sum_{j=1}^{N} \alpha_j^{opt} y_j^* x_j \right)^t x_i$$

$$= y_i^* \beta_0(\alpha^{opt}) + \sum_{j=1}^{N} \alpha_j^{opt} y_j^* y_i^* x_j x_i$$

The fact (88) that $\beta(\alpha) = \sum_{i=1}^{N} \alpha_i y_i^* x_i$ implies that only the training cases with $\alpha_i > 0$ (typically corresponding to a relatively few support vectors) determine the nature of the solution to this optimization problem. Further, for $SV$ the set indices of support vectors in the problem,

$$\|\beta(\alpha^{opt})\|^2 = \sum_{i \in SV} \sum_{j \in SV} \alpha_i^{opt} \alpha_j^{opt} y_i^* y_j^* x'_i x_j$$

$$= \sum_{i \in SV} \alpha_i^{opt} \sum_{j \in SV} \alpha_j^{opt} y_i^* y_j^* x'_i x_i$$

$$= \sum_{i \in SV} \alpha_i^{opt} (1 - y_i^* \beta_0(\alpha^{opt}))$$

$$= \sum_{i \in SV} \alpha_i^{opt}$$

the last of these following from 3. above. Then the "margin" for this problem is simply

$$M = \frac{1}{\|\beta(\alpha^{opt})\|} = \frac{1}{\sqrt{\sum_{i \in SV} \alpha_i^{opt}}}$$
11.2 The Linearly Non-separable Case

In a linearly non-separable case, the convex optimization problem (82) does not have a solution (no pair \( \beta \in \mathbb{R}^p \) and \( \beta_0 \in \mathbb{R} \) provides \( y_i^* (x_i^T \beta + \beta_0) \geq 1 \ \forall i \)). We might, therefore (in looking for good choices of \( \beta \in \mathbb{R}^p \) and \( \beta_0 \in \mathbb{R} \)) try to relax the constraints of the problem slightly. That is, suppose that \( \xi_i \geq 0 \) and consider the set of constraints

\[
y_i^* (x_i^T \beta + \beta_0) + \xi_i \geq 1 \ \forall i
\]

(the \( \xi_i \) are called "slack" variables and provide some "wiggle room" in search for a hyperplane that "nearly" separates the two classes with a good margin).

We might try to control the total amount of slack allowed by setting a limit

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

for some positive \( C \). Note that if \( y_i^* (x_i^T \beta + \beta_0) \geq 0 \) case \( i \) is correctly classified in the training set, and so if for some pair \( \beta \in \mathbb{R}^p \) and \( \beta_0 \in \mathbb{R} \) this holds for all \( i \), we have a separable problem. So any non-separable problem must have at least one negative \( y_i^* (x_i^T \beta + \beta_0) \) for any \( \beta \in \mathbb{R}^p \) and \( \beta_0 \in \mathbb{R} \) pair. This in turn requires that the budget \( C \) must be at least 1 for a non-separable problem to have a solution even with the addition of slack variables. In fact, this reasoning implies that a budget \( C \) allows for at most \( C \) mis-classifications in the training set. And in a non-separable case, \( C \) must be allowed to be large enough so that some choice of \( \beta \in \mathbb{R}^p \) and \( \beta_0 \in \mathbb{R} \) produces a classifier with training error rate no larger than \( C/N \).

In any event, we consider the optimization problem

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \| \beta \|^2 \\
\text{subject to} & \quad y_i^* (x_i^T \beta + \beta_0) + \xi_i \geq 1 \ \forall i \\
& \quad \text{for some } \xi_i \geq 0 \text{ with } \sum_{i=1}^{N} \xi_i \leq C
\end{align*}
\]

that can be thought of as generalizing the problem (82). HTF prefer to motivate (91) as equivalent to

\[
\begin{align*}
\text{maximize} & \quad M \\
\text{subject to} & \quad y_i^* (x_i^T \beta + \beta_0) \geq M (1 - \xi_i) \ \forall i \\
& \quad \text{for some } \xi_i \geq 0 \text{ with } \sum_{i=1}^{N} \xi_i \leq C
\end{align*}
\]

generalizing the original problem (80), but it’s not so clear that this is any more natural than simply beginning with (91).

A more convenient version of (91) is

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \| \beta \|^2 + C \sum_{i=1}^{N} \xi_i \\
\text{subject to} & \quad y_i^* (x_i^T \beta + \beta_0) + \xi_i \geq 1 \ \forall i \\
& \quad \text{for some } \xi_i \geq 0
\end{align*}
\]

(92)
A nice development on pages 376-378 of Izenman's book provides the following solution to this problem (92) parallel to the development in Section 11.1. Generalizing (90), the "dual" problem boils down to

$$\text{maximize } \mathbf{1}' \mathbf{\alpha} - \frac{1}{2} \mathbf{\alpha}' \mathbf{H} \mathbf{\alpha} \quad \text{subject to } 0 \leq \mathbf{\alpha} \leq C^* \mathbf{1} \text{ and } \mathbf{\alpha}' \mathbf{y}^* = 0$$

(93)

for

$$\mathbf{H} = (y_i^* y_j^* \mathbf{x}_i \mathbf{x}_j)_{N \times N}$$

(94)

The constraint $0 \leq \mathbf{\alpha} \leq C^* \mathbf{1}$ is known as a "box constraint" and the "feasible region" prescribed in (93) is the intersection of a hyperplane defined by $\mathbf{\alpha}' \mathbf{y}^* = 0$ and a "box" in the positive orthant. The $C^* = \infty$ version of this reduces to the "hard margin" separable case.

Upon solving (93) for $\mathbf{\alpha}^{\text{opt}}$, the optimal $\mathbf{\beta} \in \mathbb{R}^p$ is of the form

$$\mathbf{\beta} (\mathbf{\alpha}^{\text{opt}}) = \sum_{i \in \mathcal{SV}} \alpha_i^{\text{opt}} y_i^* \mathbf{x}_i$$

(95)

for $\mathcal{SV}$ the indices of set of "support vectors" $\mathbf{x}_i$ which have $\alpha_i^{\text{opt}} > 0$. The points with $0 < \alpha_i^{\text{opt}} < C^*$ will lie on the edge of the margin (have $\xi_i = 0$) and the ones with $\alpha_i^{\text{opt}} = C^*$ have $\xi_i > 0$. Any of the support vectors on the edge of the margin (with $0 < \alpha_i^{\text{opt}} < C^*$) may be used to solve for $\beta_0 \in \mathbb{R}$ as

$$\beta_0 (\mathbf{\alpha}^{\text{opt}}) = y_i^* - \mathbf{x}_i' \mathbf{\beta} (\mathbf{\alpha}^{\text{opt}})$$

(96)

and again, apparently for reasons of numerical stability it is common practice to average values $y_i^* - \mathbf{x}_i' \mathbf{\beta} (\mathbf{\alpha}^{\text{opt}})$ for such support vectors in order to evaluate $\beta_0 (\mathbf{\alpha}^{\text{opt}})$. Clearly, in this process the constant $C^*$ functions as a regularization/tuning parameter and large $C^*$ in (92) correspond to small $C$ in (91). Identification of a classifier requires only solution of the dual problem (93), evaluation of (95) and (96) to produce linear form (78) and classifier (79).

### 11.3 SVM's and Kernels

The form (78) is (of course and by design) linear in the coordinates of $\mathbf{x}$. A natural generalization of this development would be to consider forms that are linear in some (non-linear) functions of the coordinates of $\mathbf{x}$. There is nothing really new or special to SVM classifiers associated with this possibility if it is applied by simply defining some basis functions $h_m (\mathbf{x})$ and considering form

$$g (\mathbf{x}) = \begin{pmatrix} h_1 (\mathbf{x}) \\ h_2 (\mathbf{x}) \\ \vdots \\ h_M (\mathbf{x}) \end{pmatrix}' \mathbf{\beta} + \beta_0$$

for use in a classifier. Indeed, HTF raise this kind of possibility already in their Chapter 4, where they introduce linear classification methods. However,
the fact that in both linearly separable and linearly non-separable cases, optimal SVM’s depend upon the training input vectors $x_i$ only through their inner products (see again (89) and (94)) and our experience with RKHS’s and the computation of inner products in function spaces in terms of kernel values suggests another way in which one might employ linear forms of nonlinear functions in classification.

11.3.1 Heuristics

Let $K$ be a continuous non-negative definite kernel and consider the possibility of using functions $K(x, x_1), K(x, x_2), \ldots, K(x, x_N)$ to build new ($N$-dimensional) feature vectors

$$k(x) = \begin{pmatrix} K(x, x_1) \\ K(x, x_2) \\ \vdots \\ K(x, x_N) \end{pmatrix}$$

for any input vector $x$ (including the $x_i$ in the training set) and rather than defining inner products for new feature vectors (for input vectors $x$ and $z$) in terms of $\mathbb{R}^N$ inner products

$$k(x)^T k(z) = \sum_{k=1}^{N} K(x, x_k) K(z, x_k)$$

we instead consider using the RKHS inner products of corresponding functions

$$\langle K(x, \cdot), K(z, \cdot) \rangle_{H_K} = K(x, z)$$

Then, in place of (89) or (94) define

$$H_{N \times N} = \left( y_i^* y_j^* K(x_i, x_j) \right)$$

(97)

and let $\alpha^{opt}$ solve either (90) or (93). Then with

$$\beta(\alpha^{opt}) = \sum_{i=1}^{N} \alpha_i^{opt} y_i^* k(x_i)$$

as in the developments of the previous sections, we replace the $\mathbb{R}^N$ inner product of $\beta(\alpha^{opt})$ and a feature vector $k(x)$ with

$$\left\langle \sum_{i=1}^{N} \alpha_i^{opt} y_i^* K(x_i, \cdot), K(x, \cdot) \right\rangle_{H_K} = \sum_{i=1}^{N} \alpha_i^{opt} y_i^* \langle K(x_i, \cdot), K(x, \cdot) \rangle_{H_K} = \sum_{i=1}^{N} \alpha_i^{opt} y_i^* K(x, x_i)$$

90
Then for any $i$ for which $\alpha_i^{\text{opt}} > 0$ (an index corresponding to a support feature vector in this context) we set
\[
\beta_0(\alpha^{\text{opt}}) = y^*_i - \sum_{j=1}^{N} \alpha_j^{\text{opt}} y_j^* K(x_i, x_j)
\]
and have the analogue of (78)
\[
g(x) = \sum_{i=1}^{N} \alpha_i^{\text{opt}} y_i^* K(x, x_i) + \beta_0(\alpha^{\text{opt}}) \tag{98}
\]
with corresponding classifier
\[
f(x) = \text{sign}(g(x)) \tag{99}
\]
as an analogue of (79). It remains to argue that this classifier developed simply by analogy is a solution to any well-posed optimization problem.

### 11.3.2 An Optimality Argument

The heuristic argument for the use of kernels in the SVM context to produce form (98) and classifier (99) is clever enough that some authors simply let it stand on its own as "justification" for using "the kernel trick" of replacing $\mathbb{R}^N$ inner products of feature vectors with $\mathcal{H}_K$ inner products of basis functions. A more satisfying argument would be based on an appeal to optimality/regularization considerations. HTF’s treatment of this possibility is not helpful. The following is based on a 2002 *Machine Learning* paper of Lin, Wahba, Zhang, and Lee.

As in Section 4.4 consider the RKHS associated with the non-negative definite kernel $K$, and the optimization problem involving the "hinge loss"
\[
\min g \in \mathcal{H}_K \sum_{i=1}^{N} (1 - y_i^* (\beta_0 + g(x_i)))_+ + \lambda \frac{1}{2} \|g\|_{\mathcal{H}_K}^2 \tag{100}
\]

and $\beta_0 \in \mathbb{R}$

Exactly as noted in Section 4.4 an optimizing $g \in \mathcal{H}_K$ above must be of the form
\[
g_{\beta}(x) = \sum_{j=1}^{N} \beta_j^* K(x, x_j) = 1 \beta^* k(x)
\]

\footnote{In the discussion of AdaBoost, the exponential loss was convenient when it came time to make optimality arguments. Here, the hinge loss is convenient. For $(X, Y^*) \sim P$ it is reasonably straightforward to argue that the function $g$ minimizing the expected hinge loss $E[1 - Y^* g(X)]_+$ is $g^{\text{opt}}(x) = \text{sign}(P|Y^* = 1|X = x) - \frac{1}{2}$ so one might hope to build good classifiers as $f(x) = g^{\text{opt}}(x)$.}
so the minimization problem is

\[
\min_{\beta \in \mathbb{R}^N} \sum_{i=1}^{N} \left( 1 - y_i^* \left( \beta_0 + \beta' k(x_i) \right) \right)_+ + \frac{1}{2} \left\| \sum_{j=1}^{N} \beta_j K(x, x_j) \right\|_{\mathcal{H}_K}^2
\]

and \( \beta_0 \in \mathbb{R} \)

that is,

\[
\min_{\beta \in \mathbb{R}^N} \sum_{i=1}^{N} \left( 1 - y_i^* \left( \beta_0 + \beta' k(x_i) \right) \right)_+ + \frac{1}{2} \beta' K \beta
\]

for

\[K = (K(x_i, x_j))\]

Now apparently, this is equivalent to the optimization problem

\[
\min_{\beta \in \mathbb{R}^N} \sum_{i=1}^{N} \xi_i + \frac{1}{2} \beta' K \beta \quad \text{subject to} \quad \left\{ \begin{array}{l}
y_i^* \left( \beta_0 + \beta' k(x_i) \right) + \xi_i \geq 1 \quad \forall i \\
\text{for some } \xi_i \geq 0
\end{array} \right.
\]

and \( \beta_0 \in \mathbb{R} \)

which (also apparently) for \( H \in \mathbb{R}^{N \times N} \) as in (97) has the dual problem of the form

\[
\max \, 1' \eta - \frac{1}{2\lambda} \eta' H \eta \quad \text{subject to } 0 \leq \eta \leq 1 \quad \text{and } \eta'y^* = 0 \quad (101)
\]

or

\[
\max \, 1' \alpha - \frac{1}{2} \alpha' \left( \frac{1}{\lambda^2} H \right) \alpha \quad \text{subject to } 0 \leq \alpha \leq \lambda 1 \quad \text{and } \alpha' y^* = 0 \quad (102)
\]

That is, the function space optimization problem (100) has a dual that is the same as (93) for the choice of \( C^* = \lambda \) and kernel \( \frac{1}{\lambda} K(x, z) \) produced by the heuristic argument in Section 11.3.1. Then, if \( \eta^{\text{opt}} \) is a solution to (101), Lin et al. say that an optimal \( \beta \in \mathbb{R}^N \) is

\[
\frac{1}{\lambda} \text{diag}(y_1^*, \ldots, y_N^*) \eta^{\text{opt}}
\]

this producing coefficients to be applied to the functions \( K(\cdot, x_i) \). On the other hand, the heuristic of Section 11.3.1 prescribes that for \( \alpha^{\text{opt}} \) the solution to (102) coefficients in the vector

\[
\text{diag}(y_1^*, \ldots, y_N^*) \alpha^{\text{opt}}
\]

get applied to the functions \( \frac{1}{\lambda^2} K(\cdot, x_i) \). Upon recognizing that \( \eta^{\text{opt}} = \frac{1}{\lambda} \alpha^{\text{opt}} \) it becomes evident that for the choice of \( C^* = \lambda \) and kernel \( \frac{1}{\lambda} K(\cdot, \cdot) \), the heuristic in Section (11.3.1) produces a solution to the optimization problem (100).
11.4 Other Support Vector Stuff

Several other issues related to the kind of arguments used in the development of SVM classifiers are discussed in HTF (and Izenman). One is the matter of multi-class problems. That is, where $\mathcal{G} = \{1, 2, \ldots, K\}$ how might one employ machinery of this kind? There are both heuristic and optimality-based methods in the literature.

A heuristic one-against-all strategy might be the following. Invent 2-class problems ($K$ of them), the $k$th based on

$$ y^*_{ki} = \begin{cases} 
1 & \text{if } y^*_i = k \\
-1 & \text{otherwise}
\end{cases} $$

Then for (a single) $C^*$ and $k = 1, 2, \ldots, K$ solve the (possibly linearly non-separable) 2-class optimization problems to produce functions $g_k(x)$ (that would lead to one-versus-all classifiers $f_k(x) = \text{sign}(g_k(x))$). A possible overall classifier is then

$$ f(x) = \arg \max_{k \in \mathcal{G}} g_k(x) $$

A second heuristic strategy is to develop a voting scheme based on pair-wise comparisons. That is, one might invent \(\binom{K}{2}\) problems of classifying class $l$ versus class $m$ for $l < m$, choose a single $C^*$ and solve the (possibly linearly non-separable) 2-class optimization problems to produce functions $g_{lm}(x)$ and corresponding classifiers $f_{lm}(x) = \text{sign}(g_{lm}(x))$. For $m > l$ define $f_{ml}(x) = -f_{lm}(x)$ and define an overall classifier by

$$ f(x) = \arg \max_{k \in \mathcal{G}} \left( \sum_{m \neq k} f_{km}(x) \right) $$

or, equivalently

$$ f(x) = \arg \max_{k \in \mathcal{G}} \left( \sum_{m \neq k} I[f_{km}(x) = 1] \right) $$

Another type of question related to the support vector material is the extent to which similar methods might be relevant in regression-type prediction problems. As a matter of fact, there are loss functions alternative to squared error or absolute error that lead naturally to the use of the kind of technology needed to produce the SVM classifiers. That is, one might consider so called "$\epsilon$ insensitive" losses for prediction like

$$ L^1_\epsilon (y, \hat{y}) = \max (0, |y - \hat{y}| - \epsilon) $$

or

$$ L^2_\epsilon (y, \hat{y}) = \max (0, (y - \hat{y})^2 - \epsilon) $$

and be led to the kind of optimization methods employed in the SVM classification context. See Izenman pages 398-401 in this regard.
12 Prototype Methods and Nearest Neighbors
(Ch 13 of HTF)

We saw when looking at "linear" methods of classification in Section 3 that these can reduce to classification to the class with fitted mean "closest" in some appropriate sense to an input vector \( \mathbf{X} \). A related notion is to represent classes each by several "prototype" vectors of inputs, and to classify to the class with closest prototype. In their Chapter 13, HTF consider such classifiers and related nearest neighbor classifiers.

So consider a \( K \)-class classification problem (where \( Y^* \) takes values in \( G = \{1, 2, \ldots, K\} \)) and suppose that the coordinates of input \( \mathbf{X} \) have been standardized according training means and standard deviations.

For each class \( k = 1, 2, \ldots, K \), represent the class by prototypes \( z_{k1}, z_{k2}, \ldots, z_{KR} \) belonging to \( \mathbb{R}^p \) and consider the classifier/predictor

\[
f(\mathbf{x}) = \arg\min_k \min_{l} \| \mathbf{x} - z_{kl} \|
\]

(that is, one classifies to the class that has a prototype closest to \( \mathbf{x} \)).

The most obvious question in using such a rule is "How does one choose the prototypes?" One standard (admittedly ad hoc, but not unreasonable) method is to use the so-called "\( K \)-means (clustering) algorithm" one class at a time. (The "\( K \)" in the name of this algorithm has nothing to do with the number of classes in the present context. In fact, here the "\( K \)" naming the clustering algorithm is our present \( R \), the number of prototypes used per class. And the point in applying the algorithm is not so much to see exactly how training vectors aggregate into "homogeneous" groups/clusters as it is to find a few vectors to represent them.)

For \( T_k = \{ \mathbf{x}_i \text{ with corresponding } y_i^* = k \} \) an "\( R \)"-means algorithm might proceed by

1. randomly selecting \( R \) different elements from \( T_k \) say \( z_{k1}^{(1)}, z_{k2}^{(1)}, \ldots, z_{kR}^{(1)} \)

2. then for \( m = 2, 3, \ldots \) letting

\[
z_{kl}^{(m)} = \begin{cases} 
\text{the mean of all } \mathbf{x}_i \in T_k \text{ with } \\
\| \mathbf{x}_i - z_{kl}^{(m-1)} \| < \| \mathbf{x}_i - z_{kl'}^{(m-1)} \| \text{ for all } l \neq l' 
\end{cases}
\]

iterating until convergence.

This way of choosing prototypes for class \( k \) ignores the "location" of the other classes and the eventual use to which the prototypes will be put. A potential improvement on this is to employ some kind of algorithm (again ad hoc, but
reasonable) that moves prototypes in the direction of training input vectors in their own class and away from training input vectors from other classes. One such method is known by the name "LVQ"/"learning vector quantization." This proceeds as follows.

With a set of prototypes (chosen randomly or from an $R$-means algorithm or some other way)

$$z_{kl} \quad k = 1, 2, \ldots, K \text{ and } l = 1, 2, \ldots, R$$

in hand, at each iteration $m = 1, 2, \ldots$ for some sequence of "learning rates" $\{\epsilon_m\}$ with $\epsilon_m \geq 0$ and $\epsilon_m \searrow 0$

1. sample an $x_i$ at random from the training set and find $k, l$ minimizing $\|x_i - z_{kl}\|

2. if $y_i^* = k$ (from 1.), update $z_{kl}$ as

$$z_{kl}^{new} = z_{kl} + \epsilon_m (x_i - z_{kl})$$

and if $y_i^* \neq k$ (from 1.), update $z_{kl}$ as

$$z_{kl}^{new} = z_{kl} - \epsilon_m (x_i - z_{kl})$$

iterating until convergence.

As early as Section 1, we considered nearest neighbor methods, mostly for regression-type prediction. Consider here their use in classification problems. As before, define for each $x$ the $l$-neighborhood

$$N_l(x) = \{x_i \in \text{training set} \mid \text{closest to } x \text{ in } \mathbb{R}^p\}$$

A nearest neighbor method is to classify $x$ to the class with the largest representation in $N_l(x)$ (possibly breaking ties at random). That is, define

$$f(x) = \arg\max_k \sum_{x_i \in N_l(x)} I[y_i^* = k]$$

(103)

$l$ is a complexity parameter that might be chosen by cross-validation. Apparently, properly implemented this kind of classifier can be highly effective in spite of the curse of dimensionality. This depends upon clever/appropriate/application-specific choice of feature vectors/functions, definition of appropriate "distance" in order to define "closeness" and the neighborhoods, and appropriate local or global dimension reduction. HTF provide several attractive examples (particularly the LandSat example and the handwriting character recognition example) that concern feature selection and specialized distance measures. They then go on to consider more or less general approaches to dimension reduction for classification in high-dimensional Euclidean space.

A possibility for "local" dimension reduction is this. At $x \in \mathbb{R}^p$ one might use regular Euclidean distance to find, say, 50 neighbors of $x$ in the training
inputs to use to identify an appropriate local distance to employ in actually defining the neighborhood $\mathcal{N}_l(x)$ to be employed in (103). The following is (what I think is a correction of) what HTF write in describing a DANN (discriminant adaptive nearest neighbor) (squared) metric at $x \in \mathbb{R}^p$. Let

$$D^2(z, x) = (z - x)' Q (z - x)$$

for

$$Q = W^{-\frac{1}{2}} \left( \left( W^{-\frac{1}{2}} B W^{-\frac{1}{2}} \right)^{-1} + \epsilon I \right) W^{-\frac{1}{2}}$$

$$= W^{-\frac{1}{2}} \left( (B^*)^{-1} + \epsilon I \right) W^{-\frac{1}{2}} \tag{104}$$

for some $\epsilon > 0$ where $W$ is a pooled within-class sample covariance matrix

$$W = \sum_{k=1}^{K} \hat{\pi}_k W_k$$

$$= \sum_{k=1}^{K} \hat{\pi}_k \left( \frac{1}{n_k - 1} \sum (x_i - \bar{x}_k) (x_i - \bar{x}_k)' \right)$$

($\bar{x}_k$ is the average $x_i$ from class $k$ in the 50 used to create the local metric), $B$ is a weighted between-class covariance matrix of sample means

$$B = \sum_{k=1}^{K} \hat{\pi}_k (\bar{x}_k - \bar{x}) (\bar{x}_k - \bar{x})'$$

($\bar{x}$ is a (probably weighted) average of the $\bar{x}_k$) and

$$B^* = W^{-\frac{1}{2}} B W^{-\frac{1}{2}}$$

Notice that in (104), the "outside" $W^{-\frac{1}{2}}$ factors "sphere" $(z - x)$ differences relative to the within-class covariance structure. $B^*$ is then the between-class covariance matrix of sphered sample means. Without the $\epsilon I$, the distance would then discount differences in the directions of the eigenvectors corresponding to large eigenvalues of $B^*$ (allowing the neighborhood defined in terms of $D$ to be severely elongated in those directions). The effect of adding the $\epsilon I$ term is to limit this elongation to some degree, preventing $x_i$ "too far in terms of Euclidean distance from $x$" from being included in $\mathcal{N}_l(x)$.

A global use of the DANN kind of thinking might be to do the following. At each training input vector $x_i \in \mathbb{R}^p$, one might again use regular Euclidean distance to find, say, 50 neighbors and compute a weighted between-class-mean covariance matrix $B_i$ as above (for that $x_i$). These might be averaged to produce

$$\overline{B} = \frac{1}{N} \sum_{i=1}^{N} B_i$$

96
Then for eigenvalues of $\mathbf{B}$, say $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0$ with corresponding (unit) eigenvectors $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_p$ one might do nearest neighbor classification based on the first few features

$$v_j = \mathbf{u}_j' \mathbf{x}$$

and ordinary Euclidean distance. I’m fairly certain that this is a nearest neighbor version of the reduced rank classification idea first met in the discussion of linear classification in Section 3.