Statistical Learning

STAT 4610/5610, Fall 2023

William Kleiber[1]

1 Introduction

Statistical learning refers to a broad set of tools that can be used to explore and model complex datasets. There are overlaps with computer science and machine learning, where statistical science often places a greater emphasis on identifying and quantifying sources of *uncertainty.*

[The spam dataset consists of word frequencies and whether or not an e-mail is marked as spam]

[The CU FCQ dataset consists of course questionnaire outcomes such as instructor effectiveness, hours spent on homework, prior student interest in class]

The CU FCQ dataset consists of course questionnaire outcomes such as instructor effectiveness, hours spent on homework, prior student interest in class, etc. We might interpret hours spent on homework as an *input variable* while instructor effectiveness is an *output variable.* Our goal is to characterize a relationship between inputs and outputs.

Notation 1. *X will refer to* input variables*, which also be called* predictors, independent variables, features *or* variables*. The* output variable *will be denoted by Y, and is sometimes called the* response *or* dependent variable*.*

With multiple inputs, we distinguish between them using adorned Xs, e.g., X_1=hours spent on homework and X_2=prior interest in class.

Given p types of predictors $X_1, \ldots, X_p$, we assume there is some type of relationship

$$Y = f(X_1, \ldots, X_p) + \varepsilon$$

[1]Department of Applied Mathematics, University of Colorado, Boulder, CO. Author e-mail: `william.kleiber@colorado.edu`

where f is a fixed but unknown function of the inputs and ε is a random, mean zero error term. In this model, f represents the *systematic* information that $X_1, \ldots, X_p$ provide about Y, while ε is a mean zero error term that represents *stochastic* (or random) *unexplainable* error that cannot be explained using $X_1, \ldots, X_p$.

1.1 Typical Goals

The typical major goals are to

- Find relationships between a group of *explanatory* variables and a *response* variable that provides good predictive performance
- Interpret this relationship
- Reduce the *size* of a group of predictors for scientific or computational purposes.

Some possibilities are:

- Classification and regression
- Algorithms for large data analysis
- Recommender systems
- Spam filters
- Text processing
- Disease monitoring

Generally, most of the above reduce to estimating f in

$$Y = f(X_1, \ldots, X_p) + \varepsilon.$$

We may wish to do this for many reasons, which can be roughly categorized into *prediction* and *inference*.

Prediction

In many cases, the input variables $X_1, \dots, X_p$ are easily available (or are variables that we can control), whereas Y is quantity of main interest. If we could estimate f, say using $\hat{f}$, and we knew $X_1, \dots, X_p$, then we could *predict* Y using

$$\hat{Y} = \hat{f}(X_1, \dots, X_p).$$

For example, we might like to know if doubling homework would lead to an instructor being more effective ($\hat{f}(2X_1, X_2)$).

The accuracy of $\hat{Y}$ as a predictor for Y depends on two quantities of *reducible error* and *irreducible error*. The expected squared error between our predictor and the response is

$$\begin{aligned}
\mathbb{E}(Y - \hat{Y})^2 &= \mathbb{E}(f(X_1, \dots, X_p) + \varepsilon - \hat{f}(X_1, \dots, X_p))^2 \\
&= \mathbb{E}(f + \varepsilon - \hat{f})^2 \\
&= \mathbb{E}\big((f - \hat{f})^2 + \varepsilon^2 - 2\varepsilon(f - \hat{f})\big) \\
&= \mathbb{E}(f - \hat{f})^2 + \operatorname{Var} \varepsilon.
\end{aligned}$$

The first term is *reducible* while the second term is *irreducible*. In other words, by using more data or better learning techniques, we can improve the estimate of $\hat{f} \approx f$, whereas ε is random and cannot be predicted. For prediction, our goal will be to *minimize* this expected squared error $\mathbb{E}(Y - \hat{Y})^2$.

Inference

Inference refers to the act of estimating f and characterizing/quantifying the unpredictable error ε using a probabilistic model. Major inferential questions are:

- Which predictors are associated with the response? (Does frequency of “budget” imply an e-mail is spam?)
- What is the relationship between Y and each X_i? (Does frequency of “budget” increase or decrease likelihood an e-mail is spam?)
- Are these relationships linear or nonlinear? (Can we use $f(X) = \beta_0 + \beta_1 X$ or do we need something more complicated?)

1.2 Some Jargon

Parametric vs. Nonparametric

Learning methods for estimating f can be roughly categorized as *parametric* or *nonparametric* (or a mix between the two which is sometimes called *semiparametric*). For example, a parametric model for Y = average annual global temperature given X = average annual CO_2 concentration might use

$$f(X) = \beta_0 + \beta_1 X$$

or in other words

$$Y \approx \beta_0 + \beta_1 X.$$

The major obstacle is then to *estimate* the unknown *parameters* β_0 and β_1, for example using least squares. Once we have estimates $\hat{\beta}_0$ and $\hat{\beta}_1$, we can predict Y by

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X.$$

A nonparametric model makes no explicit assumption about the form of the relationship between Y and $X_1, \ldots, X_p$, but seeks an f that gets close to a set of training data points without being too wiggly or rough. Typically nonparametric methods are more flexible, but are not as easy to interpret as parametric models, and the modeler must weigh between these two as techniques are applied.

Supervised vs. Unsupervised

Learning techniques can be either *supervised* or *unsupervised.* Supervised techniques apply to the cases where we have a response of interest Y, given some covariates $X_1, \ldots, X_p$. Most of this class is devoted to supervised learning. Unsupervised learning applies when no response Y is available, but we are interested in learning about (for example) the *relationship* between covariates. For example, given $X_1, \ldots, X_n$, *cluster analysis* seeks to determine if the observations fall into distinct groups.

[E.g., given a set of e-mails, determine which is spam, or given a set of Netflix ratings on a shared account, how to tell which rating comes from which user]

Regression vs. Classification

A variable X (or Y) is either *quantitative*, $X \in \mathbb{R}$, or *qualitative*, $X \in \{s_1, \dots, s_m\}$. Sometimes qualitative variables are known as *categorical*. For instance, salary would be a quantitative variable whereas gender would be a categorical variable as it breaks up into *classes* or categories. Problems with a quantitative response are referred to as *regressions* while qualitative responses are known as *classifications*.

A Common Theme: Bias-Variance Tradeoff

A common way to assess the quality of fit of a model is by examining the *predictive mean squared error*. That is, for a new observation $Y = f(X) + \varepsilon$ and given an estimator $\hat{f}(X)$, we want to minimize

$$\mathbb{E}(Y - \hat{f}(X))^2 = \mathbb{E}(Y - \hat{f})^2.$$

Before we get data, $\hat{f}$ is random (because it depends on unobserved random data $Y_1, \dots, Y_n$). Assuming ε and $\hat{f}$ are uncorrelated, and that f is fixed, we have

$$\begin{aligned}
\mathbb{E}(Y - \hat{f})^2 &= \mathbb{E}(f + \varepsilon - \hat{f})^2 \\
&= \mathbb{E}(f - \hat{f})^2 + \mathbb{E}\varepsilon^2 + 2\mathbb{E}(\varepsilon(f - \hat{f})) \\
&= \mathbb{E}(f - \mathbb{E}\hat{f} + \mathbb{E}\hat{f} - \hat{f})^2 + \mathrm{Var}\varepsilon \\
&= (f - \mathbb{E}\hat{f})^2 + \mathbb{E}(\hat{f} - \mathbb{E}\hat{f})^2 + \mathrm{Var}\varepsilon \\
&= (f - \mathbb{E}\hat{f})^2 + \mathrm{Var}\hat{f} + \mathrm{Var}\varepsilon \\
&= (\text{bias of}\,\hat{f})^2 + \text{variance of}\ \hat{f} + \text{random error}
\end{aligned}$$

since $\mathbb{E}(\mathbb{E}\hat{f} - \hat{f}) = 0$.

[Picture of squared-bias/variance tradeoff with sum as MSE]

The predictive mean squared error breaks into the squared bias of $\hat{f}$, the variance of $\hat{f}$ and an irreducible and unpredictable variance of the error term. As the model for $\hat{f}$ becomes more flexible, we reduce the bias but increase the variance, whereas more rigid models exhibit smaller variance at the cost of increased bias. [Picture of MSE on y-axis as function of model complexity on x-axis with squared bias decreasing variance increasing]

2 Crash Course in Matrix Algebra

A matrix $\mathbf{A}$ with n rows and m columns (i.e., an $n \times m$ matrix) is an element of $\mathbb{R}^n \times \mathbb{R}^m$. For example, a 3×2 matrix has real-valued elements and

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix}$$

where, generally a_{ij} will refer to the (i, j)th element, that is, the row i column j element. Sometimes we shorthand this as

$$\mathbf{A} = (a_{ij})_{i=1,j=1}^{3,2} = (a_{ij})$$

A vector is just a matrix with one column, (known as a column vector), e.g.,

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}.$$

The *transpose* of a matrix is $\mathbf{A}^{\mathrm{T}} = (a_{ji})$, switching column and row places. E.g.,

$$\mathbf{A} = \begin{pmatrix} 2 & \pi \\ 0 & 10 \\ -3 & 3 \end{pmatrix} \qquad \text{has} \qquad \mathbf{A}^{\mathrm{T}} = \begin{pmatrix} 2 & 0 & -3 \\ \pi & 10 & 3 \end{pmatrix}.$$

Note that $(\mathbf{A}^{\mathrm{T}})^{\mathrm{T}} = \mathbf{A}$. A matrix is *square* if its number of rows is the number of columns. A square matrix is *symmetric* if $\mathbf{A} = \mathbf{A}^{\mathrm{T}}$, e.g.,

$$\mathbf{A} = \begin{pmatrix} 1 & 3 \\ 3 & 10 \end{pmatrix}$$

The elements $\mathrm{diag}(\mathbf{A}) = (a_{11}, a_{22}, \ldots, a_{nn})$ define the *diagonal* of a matrix. The n-dimensional *identity matrix* $\mathbf{I}$ is the square $n \times n$ matrix with 1s along the diagonal

$$\mathbf{I}_n = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

An *upper triangular matrix* is a matrix with zero entries below the main diagonal:

$$\begin{pmatrix} 2 & 8 & 3 & 0 \\ 0 & 0 & 10 & 7 \\ 0 & 0 & 4 & 7 \\ 0 & 0 & 0 & 5 \end{pmatrix}$$

and a *lower triangular matrix* has zeros above the diagonal. If we wanted to sum all elements of

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix},$$

we can write this as

$$\sum_{i=1}^{3}\sum_{j=1}^{2} a_{ij} = a_{11} + a_{12} + a_{21} + a_{22} + a_{31} + a_{32}$$

Any two $n \times p$ matrices $\mathbf{A}$ and $\mathbf{B}$ may be added by taking elementwise sums,

$$\begin{pmatrix} 2 & \pi \\ 0 & 10 \\ -3 & 3 \end{pmatrix} + \begin{pmatrix} 3 & 1 \\ 4 & 2 \\ 5 & 0 \end{pmatrix} = \begin{pmatrix} 5 & \pi + 1 \\ 4 & 12 \\ 2 & 3 \end{pmatrix}.$$

Multiplication by a constant $b\mathbf{A} = b(a_{ij}) = (ba_{ij})$ multiplies elementwise. Some properties of matrix addition: (both $\mathbf{A}$ and $\mathbf{B}$ must be $n \times m$):

- $\mathbf{A} + \mathbf{B} = \mathbf{B} + \mathbf{A}$
- $(\mathbf{A} + \mathbf{B})^{\mathrm{T}} = \mathbf{A}^{\mathrm{T}} + \mathbf{B}^{\mathrm{T}}$
- $(\mathbf{A} - \mathbf{B})^{\mathrm{T}} = \mathbf{A}^{\mathrm{T}} - \mathbf{B}^{\mathrm{T}}$
- $(\mathbf{x} + \mathbf{y})^{\mathrm{T}} = \mathbf{x}^{\mathrm{T}} + \mathbf{y}^{\mathrm{T}}$
- $(\mathbf{x} - \mathbf{y})^{\mathrm{T}} = \mathbf{x}^{\mathrm{T}} - \mathbf{y}^{\mathrm{T}}$

Matrices $\mathbf{A}(n \times m)$ and $\mathbf{B}(m \times p)$ multiply to form a $n \times p$ matrix $\mathbf{C}$ with

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj},$$

that is, take the sum of elementwise products of the ith row of $\mathbf{A}$ with the jth column of $\mathbf{B}$. E.g.,

$$\begin{pmatrix} 2 & 0 \\ 0 & 10 \\ -3 & 3 \end{pmatrix} \begin{pmatrix} 3 & 1 \\ 4 & 2 \end{pmatrix} = \begin{pmatrix} 6 & 2 \\ 40 & 20 \\ 3 & 3 \end{pmatrix}.$$

Two matrices can multiply if the number of columns of the first equals the number of rows of the second matrix, e.g., $\mathbf{AB}$ makes sense, but then $\mathbf{BA}$ may not conform, and, even if they do,

$$\mathbf{AB} \neq \mathbf{BA}.$$

Some properties of matrix multiplication:

- $\mathbf{A}(\mathbf{B}+\mathbf{C}) = \mathbf{AB}+\mathbf{AC}$
- $\mathbf{A}(\mathbf{B}-\mathbf{C}) = \mathbf{AB}-\mathbf{AC}$
- $(\mathbf{A}+\mathbf{B})\mathbf{C} = \mathbf{AC}+\mathbf{BC}$
- $(\mathbf{A}-\mathbf{B})\mathbf{C} = \mathbf{AC}-\mathbf{BC}$
- $(\mathbf{A}+\mathbf{B})(\mathbf{C}+\mathbf{D}) = \mathbf{AC}+\mathbf{BC}+\mathbf{AD}+\mathbf{BD}$
- $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$

Transposes work as

$$(\mathbf{AB})^{\mathrm{T}} = \mathbf{B}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}.$$

Note, then, that if $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^n$,

$$\mathbf{x}^{\mathrm{T}}\mathbf{y} = \sum_{i=1}^{n} x_i y_i \in \mathbb{R}.$$

which is sometimes called the dot product of $\mathbf{x}$ and $\mathbf{y}$. Moreover the squared Euclidean norm of $\mathbf{x}$ is defined as

$$\|\mathbf{x}\|_2^2 \equiv \mathbf{x}^{\mathrm{T}}\mathbf{x} = \sum_{i=1}^{n} x_i^2$$

where $\|\cdot\|_2^2$ denotes the squared L_2 length of the vector $\mathbf{x}$. More properties:

- $\sum_{i=1}^{n} \mathbf{a}^{\mathrm{T}}\mathbf{x}_i = \mathbf{a}^{\mathrm{T}} \sum_{i=1}^{n} \mathbf{x}_i$

- $\sum_{i=1}^n \mathbf{A}\mathbf{x}_i = \mathbf{A}\sum_{i=1}^n \mathbf{x}_i$
- $\sum_{i=1}^n (\mathbf{A}\mathbf{x}_i)(\mathbf{A}\mathbf{x}_i)^{\mathrm{T}} = \mathbf{A}\left(\sum_{i=1}^n \mathbf{x}_i\mathbf{x}_i^{\mathrm{T}}\right)\mathbf{A}^{\mathrm{T}}$

If $\mathbf{A}$ is square $(n \times n)$ and $\mathbf{x}$ and $\mathbf{y}$ are vectors,

$$\mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{y} = \sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j a_{ij}$$

is called a *quadratic form.* Note it is a scalar.

The *inverse* of a square matrix $\mathbf{A}$, denoted $\mathbf{A}^{-1}$, is the matrix that satisfies

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I},$$

if it exists. We have

$$(\mathbf{A}\mathbf{B})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1},$$

if all inverses exist.

A square $n \times n$ matrix $\mathbf{A}$ is *nonnegative definite* if, for any vector $\mathbf{a}$,

$$\mathbf{a}^{\mathrm{T}}\mathbf{A}\mathbf{a} \geq 0.$$

Positive definite matrices (replacing $\geq$ with $>$) *always admit an inverse.* Additionally, $\mathbf{A}$ has a "square root" called the *Cholesky factor* or *Cholesky decomposition*

$$\mathbf{A} = \mathbf{T}^{\mathrm{T}}\mathbf{T}$$

where $\mathbf{T}$ is an invertible upper triangular matrix. The Cholesky factor $\mathbf{T}$ is unique, but there are many other "square root" matrices.

The *determinant* of a matrix is a number and is written $\det(\mathbf{A}) = |\mathbf{A}|$, and has the following properties:

- $\det\mathbf{I} = 1$
- $\det(\mathbf{A}^{\mathrm{T}}) = \det\mathbf{A}$
- $\det\mathbf{A}^{-1} = 1/(\det\mathbf{A})$

- $\det(\mathbf{AB}) = \det(\mathbf{A})\det(\mathbf{B})$ for square matrices

- $\det(c\mathbf{A}) = c^n \det\mathbf{A}$ where $\mathbf{A}$ is $n \times n$.

- The determinant of a triangular or diagonal matrix is the product of its diagonal.

The *trace* of a matrix is the sum of its diagonals, $\mathrm{tr}\mathbf{A} = \sum_{i=1}^{n} a_{ii}$. We have

- $\mathrm{tr}(\mathbf{A}+\mathbf{B}) = \mathrm{tr}\mathbf{A} + \mathrm{tr}\mathbf{B}$

- $\mathrm{tr}(\mathbf{AB}) = \mathrm{tr}(\mathbf{BA})$ (even if $\mathbf{AB} \neq \mathbf{BA}$)

Two vectors $\mathbf{a}$ and $\mathbf{b}$ *orthogonal* if

$$\mathbf{a}^{\mathrm{T}}\mathbf{b} = 0.$$

If $\|\mathbf{a}\|_2 = 1$ then $\mathbf{a}$ is said to be *normalized*. Any vector can be *normalized* by

$$\mathbf{b} = \frac{\mathbf{a}}{\|\mathbf{a}\|_2}.$$

A square matrix $\mathbf{C} = (\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_p)$ is said to be *orthogonal* if its columns and rows are normalized and mutually orthogonal, whence

$$\mathbf{C}^{\mathrm{T}}\mathbf{C} = \mathbf{C}\mathbf{C}^{\mathrm{T}} = \mathbf{I}.$$

Thus, $\mathbf{C}^{-1} = \mathbf{C}^{\mathrm{T}}$. Multiplication by an orthogonal matrix has the effect of *rotating the axes*, that is, if $\mathbf{z} = \mathbf{Cx}$ then note

$$\mathbf{z}^{\mathrm{T}}\mathbf{z} = \mathbf{x}^{\mathrm{T}}\mathbf{C}^{\mathrm{T}}\mathbf{C}\mathbf{x} = \mathbf{x}^{\mathrm{T}}\mathbf{x}$$

maintains the same length.

An *eigenvalue* λ of a matrix $\mathbf{A}$ is a number where

$$\mathbf{Ax} = \lambda\mathbf{x}$$

and $\mathbf{x}$ is called the corresponding *eigenvector*. They can be calculated by finding solutions to $\det(\mathbf{A} - \lambda\mathbf{I}) = 0$. Note that, if $\mathbf{x}$ is an eigenvector, and k is a number, then

$$\mathbf{A}(k\mathbf{x}) = k\lambda(\mathbf{x}) = \lambda(k\mathbf{x})$$

so $k\mathbf{x}$ is also an eigenvector, and thus we usually scale eigenvectors to be unit length.

If λ is an eigenvalue of $\mathbf{A}$ and $\mathbf{x}$ is the corresponding eigenvector, then $1 \pm \lambda$ is an eigenvalue of $\mathbf{I} \pm \mathbf{A}$, and $\mathbf{x}$ is still the corresponding eigenvector.

If $\mathbf{A}$ is a square matrix with eigenvalues $\lambda_1, \ldots, \lambda_n$,

- $\text{tr}\mathbf{A} = \sum_{i=1}^n \lambda_i$
- $\det\mathbf{A} = \prod_{i=1}^n \lambda_i$.

Importantly,

- If $\mathbf{A}$ is positive definite, then its eigenvalues are *all positive*
- If $\mathbf{A}$ is nonnegative definite, then its eigenvalues are either positive or zero.
- If $\mathbf{A}$ is symmetric, its eigenvectors are all mutually orthogonal.

By the last property, if $\mathbf{A}$ has eigenvectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$, form

$$\mathbf{C} = [\mathbf{x}_1 \mathbf{x}_2 \cdots \mathbf{x}_n]$$

and note $\mathbf{C}$ is orthogonal. Then

$$\begin{aligned} \mathbf{A} &= \mathbf{A}\mathbf{I} \\ &= \mathbf{A}\mathbf{C}\mathbf{C}^{\mathrm{T}} \\ &= [\mathbf{A}\mathbf{x}_1 \mathbf{A}\mathbf{x}_2 \cdots \mathbf{A}\mathbf{x}_n]\mathbf{C}^{\mathrm{T}} \\ &= [\lambda_1\mathbf{x}_1 \lambda_2\mathbf{x}_2 \cdots \lambda_n\mathbf{x}_n]\mathbf{C}^{\mathrm{T}} \\ &= \mathbf{C}\mathbf{D}\mathbf{C}^{\mathrm{T}} \end{aligned}$$

where

$$\mathbf{D} = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix}$$

is diagonal with eigenvalues on the diagonal. This is known as the *spectral decomposition* or *eigendecomposition* of $\mathbf{A}$. We can also *diagonalize* $\mathbf{A}$ by

$$\mathbf{C}^{\mathrm{T}}\mathbf{A}\mathbf{C} = \mathbf{D}.$$

A *square root matrix* is $\mathbf{A}^{1/2} = \mathbf{C}\mathbf{D}^{1/2}\mathbf{C}^{\mathrm{T}}$, where $\mathbf{D}^{1/2} = \mathrm{diag}(\lambda_1^{1/2}, \ldots, \lambda_n^{1/2})$. The square of $\mathbf{A}$ can be calculated

$$\mathbf{A}^2 = \mathbf{C}\mathbf{D}^2\mathbf{C}^{\mathrm{T}}$$

and the inverse

$$\mathbf{A}^{-1} = \mathbf{C}\mathbf{D}^{-1}\mathbf{C}^{\mathrm{T}}$$

all of which only involve applying these functions to the eigenvalues in $\mathbf{D}$, which is one reason the eigendecomposition is nice.

The *rank* of a matrix is the number of linearly independent column vectors (or row vectors). If $\mathbf{A}$ is $n \times p$, then

- $\mathrm{rank}(\mathbf{A}) \leq \min(n, p)$
- $\mathrm{rank}(\mathbf{A}^{\mathrm{T}}\mathbf{A}) = \mathrm{rank}(\mathbf{A}\mathbf{A}^{\mathrm{T}}) = \mathrm{rank}(\mathbf{A})$
- If $\mathbf{A}$ is $n \times n$ and has full rank (n) then $\mathbf{A}$ is invertible.

We will mostly be only dealing with full rank matrices.

Let $\mathbf{A}$ be an $n \times p$ matrix of rank k. The *singular value decomposition* of $\mathbf{A}$ is

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathrm{T}}$$

where $\mathbf{U}$ is $n \times k$, $\mathbf{D}$ is $k \times k$ and $\mathbf{V}$ is $p \times k$. $\mathbf{D} = \mathrm{diag}(\lambda_1, \ldots, \lambda_k)$ contains the positive square roots of the $\lambda_1^2, \ldots, \lambda_k^2$ nonzero eigenvalues of $\mathbf{A}\mathbf{A}^{\mathrm{T}}$ or $\mathbf{A}^{\mathrm{T}}\mathbf{A}$. The k columns of $\mathbf{U}$ are normalized eigenvectors of $\mathbf{A}\mathbf{A}^{\mathrm{T}}$ corresponding to eigenvalues $\lambda_1^2, \ldots, \lambda_k^2$. The k columns of $\mathbf{V}$ are the normalized eigenvectors of $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ corresponding to eigenvalues $\lambda_1^2, \ldots, \lambda_k^2$. Then,

$$\mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{V}^{\mathrm{T}}\mathbf{V} = \mathbf{I}.$$

If $\mathbf{A}$ is positive definite, then the singular value decomposition is the same as the spectral decomposition.

3 Random Vectors

Recall the definition of covariance and correlation: if X and Y are random variables with mean and standard deviation μ_X, σ_X and μ_Y, σ_Y, respectively, then

$$\operatorname{Cov}(X,Y) := \mathbb{E}\big((X-\mu_X)(Y-\mu_Y)\big)$$

and

$$\operatorname{Cor}(X,Y) := \frac{\operatorname{Cov}(X,Y)}{\sigma_X\sigma_Y} \quad (=\rho).$$

In particular, $\operatorname{Cor}(X,Y) \in [-1,1]$ is unitless and is a measure of the *linear dependence* between X and Y, and $\operatorname{Cov}(X,Y)$ is in the units of X times the units of Y. Covariances enjoy the following properties:

- $\operatorname{Var}X = \operatorname{Cov}(X,X)$
- $\operatorname{Cov}(aX+b,Y) = a\operatorname{Cov}(X,Y)$ for any $a,b\in\mathbb{R}$ [prove]
- $\operatorname{Cov}(X,Y) = \operatorname{Cov}(Y,X)$
- $\operatorname{Cov}(X+Y,Z) = \operatorname{Cov}(X,Z) + \operatorname{Cov}(Y,Z)$.

How are covariances defined for vectors of random variables? Suppose $\mathbf{X} = (X_1,\ldots,X_n)^{\mathrm{T}}$ and $\mathbf{Y} = (Y_1,\ldots,Y_m)^{\mathrm{T}}$ are two random vectors, where $^{\mathrm{T}}$ denotes the transpose. Define $\operatorname{Cov}(\mathbf{X},\mathbf{Y})$ to be the $n\times m$ matrix with (i,j)th entry $\operatorname{Cov}(X_i,Y_j)$, that is,

$$\operatorname{Cov}(\mathbf{X},\mathbf{Y}) = \begin{pmatrix} \operatorname{Cov}(X_1,Y_1) & \operatorname{Cov}(X_1,Y_2) & \cdots & \operatorname{Cov}(X_1,Y_m) \\ \operatorname{Cov}(X_2,Y_1) & \operatorname{Cov}(X_2,Y_2) & \cdots & \operatorname{Cov}(X_2,Y_m) \\ \vdots & \vdots & \ddots & \vdots \\ \operatorname{Cov}(X_n,Y_1) & \operatorname{Cov}(X_n,Y_2) & \cdots & \operatorname{Cov}(X_n,Y_m) \end{pmatrix}$$

The following properties also hold for covariances of random vectors:

- $\operatorname{Var}\mathbf{X} = \operatorname{Cov}(\mathbf{X},\mathbf{X})$ [This is how we define the variance of a random vector]
- $\operatorname{Cov}(A\mathbf{X}+\boldsymbol{\mu}, B\mathbf{Y}+\boldsymbol{\nu}) = A\operatorname{Cov}(\mathbf{X},\mathbf{Y})B^{\mathrm{T}}$ for any $k\times n$ matrix A, $j\times m$ matrix B, $\boldsymbol{\mu}\in\mathbb{R}^k$ and $\boldsymbol{\nu}\in\mathbb{R}^j$

- $\text{Cov}(\mathbf{X}, \mathbf{Y}) = \text{Cov}(\mathbf{Y}, \mathbf{X})^{\mathrm{T}}$.

If $\mathbf{X}$ is a random vector such that $\text{Var}X_i = \sigma_i^2$, then the covariance matrix $\text{Cov}(\mathbf{X}, \mathbf{X}) = (\text{Cov}(X_i, X_j))_{i,j=1}^n$ can be transformed into a *correlation matrix*

$$\text{Cor}(\mathbf{X}, \mathbf{X}) = \left(\frac{\text{Cov}(X_i, X_j)}{\sigma_i \sigma_j}\right)_{i,j=1}^n .$$

The correlation matrix is usually easier to interpret since it consists of all pairwise correlations between the component random variables. Note the diagonal of $\text{Cor}(\mathbf{X}, \mathbf{X})$ is 1s.

The most relevant example we will be faced with is actually fairly straightforward: suppose we have a set of uncorrelated, mean zero random variables $\varepsilon_1, \ldots, \varepsilon_n$, each with variance $\text{Var}\varepsilon_i = \sigma^2$. Then, if

$$\boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}$$

we have

$$\text{Var}\boldsymbol{\varepsilon} = \begin{pmatrix} \sigma^2 & 0 & \cdots & 0 \\ 0 & \sigma^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma^2 \end{pmatrix} = \sigma^2 I$$

because $\text{Cov}(\varepsilon_i, \varepsilon_j) = 0$ for $i \neq j$.

4 Linear Regression

[R example (IntroMultipleRegression.R)]

Linear regression is a simple but powerful approach for supervised learning. Given a set of predictors (or features or covariates or explanatory variables) $X_1, \ldots, X_p$ and a response (or supervisor) Y, we should be able to answer the following questions:

- Is there a relationship between Y and X_i? If so, how strong is it?
- How accurately can we quantify a relationship?
- How accurately can we predict Y, given a set of covariates?
- Is the relationship linear, and are there interactions between $X_1, \ldots, X_p$?

4.1 Simple Linear Regression

In the simple linear regression case, we have a single quantitative response Y and a single predictor X. We assume that, up to some random error, Y and X share a linear relationship,

$$Y = \beta_0 + \beta_1 X + \varepsilon \tag{1}$$

where ε is the random error term. Sometimes we say Y is being *regressed on* X. This is our first example of a *statistical model.* Here, β_0 and β_1 are unknown *coefficients* or *parameters* which must be estimated. This is the typical model with $f(X) = \beta_0 + \beta_1 X$. This function is known as the *population regression line*, and ε is the *residual.* Of course, β_0 is the estimated value of Y for $X = 0$ and β_1 is the increase in Y for a unit increase of X.

In practice we have a set of n *observation pairs*

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$$

of both covariates x_i and responses y_i. If the relationship (1) holds, then we have a set of observation equations

$$\begin{aligned} y_1 &= \beta_0 + \beta_1 x_1 + \varepsilon_1 \\ &\vdots \\ y_n &= \beta_0 + \beta_1 x_n + \varepsilon_n \end{aligned}$$

where the ε_i are now fixed numbers. Equivalently, a priori these are random variables so

$$Y_i = \beta_0 + \beta_1 x_i + \varepsilon_i, \quad i = 1, \ldots, n \tag{2}$$

where ε_i are random.

[Picture]

4.2 Multiple Linear Regression

Sometimes it makes sense to regress Y on multiple covariates $X_1, X_2, \ldots, X_p$. In this case the *multiple linear regression* model assumes

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon \tag{3}$$

with the only difference here being the introduction of extra covariates. If we have n observations, $y_1, \ldots, y_n$, where the ith has p corresponding covariates $x_{i1}, \ldots, x_{ip}$, then

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i \tag{4}$$

where $\{\varepsilon_i\}$ are random.

Note the interpretation of β_i *is different* here: for a unit increase in X_i, β_i is the average increase in Y *with all other covariates held fixed.*

The model for $y_1, \ldots, y_n$ can be written in matrix notation. Define

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix}, \quad \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix} \quad \mathbf{X} = \begin{pmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix},$$

Note $\boldsymbol{\beta}$ is $(p+1) \times 1$ and $\mathbf{X}$ is $n \times (p+1)$. The first column of 1s in $\mathbf{X}$ is for the β_0 term. We can then write the model for all observations $\mathbf{y}$ succinctly as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}.$$

Sometimes it's convenient to write

$$\mathbf{x}_i = \begin{pmatrix} 1 \\ x_{i1} \\ \vdots \\ x_{ip} \end{pmatrix}, \quad \text{so that} \quad y_i = \mathbf{x}_i^{\mathrm{T}}\boldsymbol{\beta} + \varepsilon_i \quad \text{and} \quad \mathbf{X} = \begin{pmatrix} \mathbf{x}_1^{\mathrm{T}} \\ \mathbf{x}_2^{\mathrm{T}} \\ \vdots \\ \mathbf{x}_n^{\mathrm{T}} \end{pmatrix}.$$

4.3 Assumptions

There are differing levels of assumptions that are typically made in practice regarding (4).

A1
1. The relationship (3) holds
2. ε_i are iid (independent and identically distributed) $N(0, \sigma^2)$ for all $i = 1, \ldots, n$

(i.e. $Y \sim N(\boldsymbol{\beta}^{\mathrm{T}}\mathbf{x}, \sigma^2)$)

A2
1. The relationship (3) holds
2. $\mathbb{E}\varepsilon_i = 0$
3. $\mathrm{Var}\varepsilon_i = \sigma^2$ (homoskedasticity)
4. ε_i and ε_j are iid

A3
1. The relationship (3) holds
2. $\mathbb{E}\varepsilon_i = 0$
3. $\mathrm{Var}\varepsilon_i = \sigma^2$ (homoskedasticity)
4. ε_i and ε_j are uncorrelated for $i \neq j$

A4
1. The relationship (3) holds
2. $\mathbb{E}\varepsilon_i = 0$
3. $\mathrm{Var}\varepsilon_i < \infty$.

[Picture of A1: what would we expect plot to look like with $n \gg 0$?]

What is common to all levels of assumptions is that the posited relationship (1) (or (3)) holds. The differences between the three assumptions are about the assumed behavior of the residuals. A1 is the strongest in that we explicitly assume the residuals are normally distributed, whereas A2 relaxes this assumption to being iid from some unspecified probability distribution. Statistical independence is a stronger statement than correlation, and A4 is the weakest in that we only put assumptions on the first two *moments* of the variables. For instance, $\mathbb{E}\varepsilon_1^3$ may be different than $\mathbb{E}\varepsilon_2^3$ in A3, but not for A2 or A1.

The fourth assumption is the weakest, and relaxes the *homoskedasticity* property (equal variance among all residuals) by allowing for *heteroskedasticity* (nonconstant variance). We will work with A1. Our goals are usually

1. Estimate parameters $\boldsymbol{\beta}$ and σ^2 (and quantify uncertainty in our estimates)
2. Assess the validity of our assumed model (3)
3. Predict response at a new covariate value $\mathbf{x} = \mathbf{x}_*$.

For our observation model (4), any of assumptions A1-A3 imply that

$$\mathbb{E}\boldsymbol{\varepsilon} = \mathbf{0} \quad \text{and} \quad \text{Var}\boldsymbol{\varepsilon} = \text{Cov}(\boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}) = \sigma^2 \mathbf{I}_n,$$

which will be our working assumptions for the remainder of the class.

4.4 Ordinary least squares

The heuristic for estimating the regression parameters β_0 and β_1 is that we should minimize the distance between y_i and its fitted value $\beta_0 + \beta_1 x_i$.

[Picture of data with candidate lines]

Given observations $\mathbf{y}$ from the model

$$\mathbf{Y} = \boldsymbol{\beta}\mathbf{X} + \boldsymbol{\varepsilon}$$

we have the OLS estimator for $\boldsymbol{\beta}$ as minimizing

$$\sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}))^2 = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^{\mathrm{T}} (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}),$$

which results in

$$\hat{\boldsymbol{\beta}}_{OLS} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}.$$

[Check dimensions] Note this is *not* the best way to compute $\hat{\boldsymbol{\beta}}$; in practice most programs use a QR decomposition that is more robust against rounding errors.

The OLS estimators are *unbiased*:

$$\begin{aligned}
\mathbb{E}\hat{\boldsymbol{\beta}} &= \mathbb{E}(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{Y} \\
&= (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbb{E}\mathbf{Y} \\
&= (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{X}\boldsymbol{\beta} \quad \text{[why?]} \\
&= \boldsymbol{\beta}.
\end{aligned}$$

Moreover,

$$\begin{aligned}
\mathrm{Var}\hat{\boldsymbol{\beta}} &= \mathrm{Var}(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{Y} \\
&= (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathrm{Var}(\mathbf{Y},\mathbf{Y})\mathbf{X}(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1} \\
&= \sigma^2(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{X}(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1} \quad \text{[why?]} \\
&= \sigma^2(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}.
\end{aligned}$$

[note this is a variance-covariance matrix]

Standard errors for the OLS estimator $\hat{\boldsymbol{\beta}}_{OLS}$ are found by taking the square root of the diagonal of

$$\widehat{\mathrm{Var}}(\hat{\boldsymbol{\beta}}_{OLS}) = \hat{\sigma}^2(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}$$

where the residual variance is estimated by

$$\hat{\sigma}^2 = \frac{(\mathbf{y}-\mathbf{X}\hat{\boldsymbol{\beta}})^{\mathrm{T}}(\mathbf{y}-\mathbf{X}\hat{\boldsymbol{\beta}})}{n-(p+1)}.$$

This is an unbiased estimator for σ^2 (but is *not* the MLE). We can also write the residual sum of squares in a few formats:

$$\begin{aligned}
RSS &= (\mathbf{y}-\mathbf{X}\hat{\boldsymbol{\beta}})^{\mathrm{T}}(\mathbf{y}-\mathbf{X}\hat{\boldsymbol{\beta}}) \\
&= \mathbf{y}^{\mathrm{T}}\mathbf{y} - \hat{\boldsymbol{\beta}}^{\mathrm{T}}(\mathbf{X}^{\mathrm{T}}\mathbf{X})\hat{\boldsymbol{\beta}} \\
&= \mathbf{y}^{\mathrm{T}}\mathbf{y} - \hat{\boldsymbol{\beta}}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{y}.
\end{aligned}$$

Under A1,

$$\frac{n-(p+1)}{\sigma^2}\hat{\sigma}^2 \sim \chi^2_{n-(p+1)}.$$

[R example (IntroLinearRegression.R)]

Confidence Intervals

An aside: If $X \sim N(\mu, \sigma^2)$, note that

$$P\left(-1.96 \le \frac{X-\mu}{\sigma} \le 1.96\right) \approx 0.9500042 \approx 95\%$$

so,

$$P\left(\mu - 1.96\sigma \le X \le \mu + 1.96\sigma\right) \approx 95\%$$

or,

$$P\left(X - 1.96\sigma \le \mu \le X + 1.96\sigma\right) \approx 95\%.$$

The middle statement says that $\mu \pm 1.96\sigma$ contains X 95% of the time. The last statement says that $X \pm 1.96\sigma$ will contain μ 95% of the time, and is the basis for constructing a *confidence interval.* Sometimes this is approximated as $X \pm 2\sigma$. [Note one of these is a *fixed* interval while the other is *random*].

Let's simplify notation just a bit. Call

$$C = \frac{1}{n} + \frac{\overline{x}^2}{\sum_{i=1}^n (x_i - \overline{x})^2}$$

so that

$$\hat{\beta}_0 \sim N(\beta_0, \sigma^2 C).$$

Suppose we knew σ^2. We call the following an *exact* 95% *confidence interval for* β_0

$$\begin{aligned}&\left(\hat{\beta}_0 - 1.96\sqrt{\text{Var}\hat{\beta}_0}, \hat{\beta}_0 + 1.96\sqrt{\text{Var}\hat{\beta}_0}\right)\\ &= \left(\hat{\beta}_0 - 1.96\sigma\sqrt{C}, \hat{\beta}_0 + 1.96\sigma\sqrt{C}\right).\end{aligned}$$

This would be the interval that contains β_0 95% of the time, or, in other words, if we repeated the experiment 1000 times, we would expected this interval to contain β_0 950 of those times. We don't know σ^2, however, so we must rely on a plug-in estimator $\hat{\sigma}^2$ when constructing such an interval.

Two options:

Option 1 Use

$$\hat{\beta}_0 \pm 1.96\hat{\sigma}\sqrt{C} \qquad \text{or} \qquad \hat{\beta}_0 \pm 2\hat{\sigma}\sqrt{C}$$

as an *approximate* 95% *confidence interval for* β_0.

Option 2 Be more careful.

Recall that a t random variable with r degrees of freedom can be represented as

$$T = \frac{X}{\sqrt{Y/r}} =_d \frac{N(0,1)}{\sqrt{\chi^2_r/r}}$$

where $X \sim N(0,1)$ and $Y \sim \chi^2_r$ are independent random variables and where we write $T \sim t_r$. [Picture of t_r distribution compared to normal] Under A1, $\hat{\beta}_0 \sim N(\beta_0, \sigma^2 C)$ is *exact*. Thus,

$$\frac{\hat{\beta}_0 - \beta_0}{\sigma\sqrt{C}} \sim N(0,1)$$

and

$$\frac{\hat{\beta}_0 - \beta_0}{\hat{\sigma}\sqrt{C}} =_d \frac{N(0,\sigma^2)}{\hat{\sigma}} =_d \frac{N(0,\sigma^2)}{\sqrt{\hat{\sigma}^2}} =_d \frac{\sigma N(0,1)}{\sqrt{\hat{\sigma}^2}} =_d \frac{N(0,1)}{\sqrt{\frac{\hat{\sigma}^2}{\sigma^2}}} =_d \frac{N(0,1)}{\sqrt{\frac{(n-2)\hat{\sigma}^2}{(n-2)\sigma^2}}}$$

$$=_d \frac{N(0,1)}{\sqrt{\chi^2_{n-2}/(n-2)}} =_d t_{n-2}$$

The last step requires a little extra care since it's not obvious that $\hat{\beta}_0$ and $\hat{\sigma}^2$ are independent.

Now, this implies

$$P\left(t_{n-2}(0.025) \le \frac{\hat{\beta}_0 - \beta_0}{\hat{\sigma}\sqrt{C}} \le t_{n-2}(0.975)\right) = 0.95$$

where $t_{n-2}(0.025)$ is the 2.5% quantile of a t_{n-2} distribution. [Note that, by symmetry, $t_{n-2}(0.025) = -t_{n-2}(0.975)$]

Option 2 Under A1, an *exact* 95% *confidence interval for* β_0 is

$$\hat{\beta}_0 \pm t_{n-2}(0.025)\hat{\sigma}\sqrt{C}$$

Table 1: Values of $t_{n-2}(0.975)$ for various n.

$n-2$	10	20	50	100	200
$t_{n-2}(0.975)$	2.23	2.09	2.01	1.98	1.97

Note that, in general if we wanted a $100(1-\alpha)\%$ confidence interval, we would use

$$\hat{\beta}_0 \pm t_{n-2}(\alpha/2)\hat{\sigma}\sqrt{C}$$
$$\big(\pm t_{n-2}(1-\alpha/2)\hat{\sigma}\sqrt{C}\big)$$

where $t_{n-2}(\alpha/2)$ is the lower $\alpha/2 \times 100\%$ quantile of the t_{n-2} distribution. [e.g., if $\alpha = 0.05$, then $100(1-\alpha)\% = 95\%$] Table 1 gives some typical values of $t_{n-2}(0.975)$. [Note that small n implies we end up using wider confidence intervals, i.e., we exhibit less confidence if we don't approximate by using normal cutoffs]

The α level controls the Type I error – that is, the frequency of which the CI does *not* contain the truth. E.g., $\alpha = 0.05$, then 95% of experiments' CIs will contain the true parameters. [This is no guarantee that *this* experiment's CI does!]

In summary, 95% approximate CIs for our estimated parameters are

$$\hat{\beta}_0 \pm 1.96SE(\hat{\beta}_0)$$
$$\hat{\beta}_1 \pm 1.96SE(\hat{\beta}_1)$$

while exact 95% CIs are

$$\hat{\beta}_0 \pm t_{n-2}(0.025)SE(\hat{\beta}_0)$$
$$\hat{\beta}_1 \pm t_{n-2}(0.025)SE(\hat{\beta}_1)$$

where $SE = \sqrt{\widehat{\mathrm{Var}}}$ are standard errors, that is, *estimated* standard deviations. In particular,

$$\widehat{\mathrm{Var}}(\hat{\beta}_1) = \frac{\hat{\sigma}^2}{\sum_{i=1}^n (x_i - \overline{x})^2}$$
$$\widehat{\mathrm{Var}}(\hat{\beta}_0) = \frac{\hat{\sigma}^2}{n} + \frac{\hat{\sigma}^2\overline{x}^2}{\sum_{i=1}^n (x_i - \overline{x})^2}.$$

The standard error attempts to quantify an answer to the question *how certain are we about the value of the estimator*?

Prediction

Given a linear model

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}.$$

Suppose we want to predict Y_* for new set of covariates $\mathbf{x}_* = (1, x_{*1}, x_{*2}, \ldots, x_{*p})^{\mathrm{T}}$. There are two we might want to predict with uncertainty at $\mathbf{x}_*$:

- The mean value $\mathbf{x}_*^{\mathrm{T}}\boldsymbol{\beta}$ (fit)
- A single new observation $y_* = \mathbf{x}_*^{\mathrm{T}}\boldsymbol{\beta} + \varepsilon_*$ (prediction).

[In the first case we want to quantify our uncertainty about the mean function, whereas in the second case we want to include our uncertainty about what the residual will be]

The natural point predictor for *both cases* is

$$\hat{y}_* = \mathbf{x}_*^{\mathrm{T}}\hat{\boldsymbol{\beta}} = \mathbf{x}_*^{\mathrm{T}}(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}\mathbf{y}.$$

[Note predictor is linear in the observations $\mathbf{y}$!]

Our uncertainty depends on which case we're looking at. Recall we define $SE(\cdot) = \sqrt{\widehat{\mathrm{Var}}(\cdot)}$. Note

$$\mathrm{Var}(\mathbf{x}^{\mathrm{T}}\hat{\boldsymbol{\beta}}) = \mathbf{x}^{\mathrm{T}}\mathrm{Var}(\hat{\boldsymbol{\beta}})\mathbf{x}.$$

The standard error of prediction is then

$$\begin{aligned}
\widehat{\mathrm{Var}}(\mathbf{x}_*^{\mathrm{T}}\hat{\boldsymbol{\beta}}) &= \hat{\sigma}^2\mathbf{x}_*^{\mathrm{T}}(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{x}_* && \text{(fit/confidence)} \\
\widehat{\mathrm{Var}}(\mathbf{x}_*^{\mathrm{T}}\hat{\boldsymbol{\beta}} + \varepsilon_*) &= \hat{\sigma}^2(1 + \mathbf{x}_*^{\mathrm{T}}(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{x}_*) && \text{(prediction)}
\end{aligned}$$

and 95% predictive confidence intervals are then

$$\begin{aligned}
&\hat{y}_* \pm t_{n-(p+1)}(\alpha/2)SE(\mathbf{x}_*^{\mathrm{T}}\hat{\boldsymbol{\beta}}) && \text{(fit/confidence)} \\
&\hat{y}_* \pm t_{n-(p+1)}(\alpha/2)SE(\mathbf{x}_*^{\mathrm{T}}\hat{\boldsymbol{\beta}} + \varepsilon_*) && \text{(prediction)}
\end{aligned}$$

under A1. [Note the standard error of prediction is *greater* than just $\hat{\sigma}$, since there is uncertainty in estimating $\boldsymbol{\beta}$ – follows since $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ is nonnegative definite]. If the residuals $\boldsymbol{\varepsilon}$ are correlated, then we can improve the point and interval estimates by taking account of this extra structure, but this is left for another class.

4.5 Diagnostics

Based on our estimates $\hat{\boldsymbol{\beta}}$, we have *fitted values*

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$$

and *estimated residuals*

$$\hat{\boldsymbol{\varepsilon}} = \mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}.$$

[Picture of estimated residuals vs true residuals] If our assumptions are correct, then the estimated residuals $\hat{\boldsymbol{\varepsilon}}$ should not have any structure.

One important diagnostic plot is the residuals vs. fitted values scatterplot, which can indicate a number of assumption violations:

1. If there are definite trends in the plot then the assumed linear relationship may be violated. [Picture]
2. If the variability changes with fitted values, then homoskedasticity may be violated; this latter case is when the errors exhibit *heteroskedasticity*. [Picture]
3. A few unusually large residuals may be evidence of *outliers*. Outliers won't (necessarily) change $\hat{\beta}_i$, but they do inflate $\hat{\sigma}^2$. [Picture]

The strongest set of assumptions, A1, suggest the residual terms ε_i are normally distributed. To assess normality of the estimated residuals $\{\hat{\varepsilon}_i\}_i$, the *quantile-quantile plot* (Q-Q plot) is often used. The Q-Q plot plots the theoretical quantiles of a standard normal versus the estimated quantiles of the standardized observed residuals. If the plot falls along the identity line, it may be reasonable to assume the errors arose from a normal distribution. Q-Q plots can be used to assess heavy-tailedness and skewness compared to the normal distribution. [Picture of standard and heavy-tailed Q-Q plot]

A fundamental quantity in regression (and beyond) is the *hat matrix*,

$$\mathbf{H} = \mathbf{X}(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}.$$

First note

$$\mathbf{H}^2 = \mathbf{H}\mathbf{H} = \mathbf{H}$$

so $\mathbf{H}$ is idempotent. The *fitted values* are

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{X}(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} = \mathbf{H}\mathbf{y},$$

so the hat matrix converts observations into fitted values. The estimated residuals are

$$\hat{\varepsilon}_i = y_i - \hat{y}_i$$

which we can write as

$$\begin{aligned}\hat{\boldsymbol{\varepsilon}} &= \mathbf{y} - \hat{\mathbf{y}} \\ &= \mathbf{y} - \mathbf{H}\mathbf{y} \\ &= (\mathbf{I} - \mathbf{H})\mathbf{y}.\end{aligned}$$

Thus,

$$\begin{aligned}\mathrm{Var}\hat{\boldsymbol{\varepsilon}} &= \mathrm{Cov}(\hat{\boldsymbol{\varepsilon}}, \hat{\boldsymbol{\varepsilon}}) \\ &= (\mathbf{I} - \mathbf{H})\mathrm{Cov}(\boldsymbol{\varepsilon}, \boldsymbol{\varepsilon})(\mathbf{I} - \mathbf{H})^{\mathrm{T}} \\ &= \sigma^2(\mathbf{I} - \mathbf{H}).\end{aligned}$$

[We need to use $\mathbf{H}^2 = \mathbf{H}$ for this] Thus, the standard error for $\hat{\varepsilon}_i$ is

$$SE(\hat{\varepsilon}_i) = \hat{\sigma}\sqrt{1 - H_{ii}}$$

where H_{ii} is the ith diagonal entry of $\mathbf{H}$. Under A1, the *studentized residuals*

$$\hat{\varepsilon}_i^* = \frac{\hat{\varepsilon}_i - 0}{SE(\hat{\varepsilon}_i)} \approx N(0, 1).$$

This gives us a method to look for *outliers*, where we might suspect a particular y_i is an outlier if $\hat{\varepsilon}_i^*$ is smaller than -2 or larger than 2.

Now note

$$\hat{y}_i = H_{i1}y_1 + \cdots + H_{ii}y_i + \cdots + H_{in}y_n,$$

so H_{ii} is the influence of the observation y_i on its own fitted value. We define H_{ii} to be the *leverage* of y_i, a measure of its *potential* for being influential on the final fit, but note that

H_{ii} only depends on $\mathbf{X}$, and not the actual value of y_i. To measure the the actual influence of a data point y_i, we use *Cook's distance (Cook's D)* as

$$D_i = \frac{H_{ii}}{p(1-H_{ii})}(\hat{\varepsilon}_i^*)^2.$$

High values of leverage imply *potential influence* on the model fit while high values of Cook's D imply *actual influence* on the model fit.

Assessing Quality of Model Fit

Given two competing models (e.g., one with/one without a particular covariate), how do we decide which is better? That is, how do we quantify the *goodness-of-fit*? Recall that

$$RSS = (\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}) = \sum_{i=1}^{n}(y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_p x_{ip}))^2$$

and

$$r^2 = 1 - \frac{RSS}{SYY}$$

We could use the model that has a better (lower RSS or higher r^2) value, except that these will *always* decrease/increase respectively with the addition of new covariates. Thus, we need an approach that *rewards* model fit while *penalizing* complexity.

If we entertain a model with d covariates $X_1, \ldots, X_d$ fit to n observations, then we define *Mallow's C_p* as

$$C_p = \frac{1}{n}(RSS + 2d\hat{\sigma}^2).$$

The second term typically increases as d increases, and we favor models with the smallest C_p value. [Note the book gives a different version of C_p which some people use in practice].

If a model is fit by maximum likelihood, then *Akaike's information criterion* (AIC) or the *Bayesian information criterion* (BIC) may be used. [Recall that OLS is equivalent to ML if errors are normal]. Up to a constant, AIC is

$$AIC = \frac{1}{n\hat{\sigma}^2}(RSS + 2d\hat{\sigma}^2)$$

so, for least squares models, C_p and AIC are proportional (i.e., they agree on the best model). Up to a constant, BIC is

$$BIC = \frac{1}{n}(RSS + \log(n)d\hat{\sigma}^2).$$

Note that since $\log n > 2$ for any $n > 7$, BIC tends to place more penalty on models with many variables d. Under BIC, the best model is that with the smallest BIC value.

Finally, the *adjusted* r^2 statistic is defined by

$$\text{Adjusted } r^2 = 1 - \frac{RSS/(n-(d+1))}{TSS/(n-1)}$$

The best model is that that *maximizes* the adjusted r^2. Intuition: once all relevant variables are in the model, RSS is relatively stable, and increasing d should be penalized (note that if RSS constant then the adjusted r^2 is decreasing in d).

[R example (Regression.R)]

4.6 Potential Issues

In multiple regression there are numerous potential issues to consider.

Categorical (Qualitative) Predictors

So far we've dealt with *quantitative* predictors, e.g., $X \in \mathcal{D} \subseteq \mathbb{R}$. *Categorical* or *qualitative* covariates only take on *finitely* many values (also sometimes called *factors*).

For example, if x_i is gender, then it has only two *levels*. We need to choose a convention to code it numerically using an indicator or *dummy variable* that takes on two possible values, e.g.,

$$x_i = \begin{cases} 1 & \text{if } i\text{th person is female} \\ 0 & \text{if } i\text{th person is male} \end{cases}$$

What happens to a simple model in this case?

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i = \begin{cases} \beta_0 + \beta_1 + \varepsilon_i & \text{if } i\text{th person is female} \\ \beta_0 + \varepsilon_i & \text{if } i\text{th person is male} \end{cases}$$

Now β_0 is *the average outcome for males*, while $\beta_0 + \beta_1$ is the average outcome for females. Another interpretation is that β_1 is the average difference between females and males. The coefficient can be interpreted as a *shift in the mean* for $x_i = 1$. If we coded this differently, say

$$x_i = \begin{cases} 1 & \text{if } i\text{th person is female} \\ -1 & \text{if } i\text{th person is male} \end{cases}$$

then

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i = \begin{cases} \beta_0 + \beta_1 + \varepsilon_i & \text{if } i\text{th person is female} \\ \beta_0 - \beta_1 + \varepsilon_i & \text{if } i\text{th person is male} \end{cases}$$

Now β_0 is the *overall average* (ignoring gender), and β_1 is the amount females are above (and males are below) the average. Note that predictions will always be the same, regardless of the coding of x_i, but the *interpretation* will change.

If there are more than two levels, then we use additional dummy variables. For example, if Y is life expectancy measured in each country of the world, and countries are grouped into *Africa*, *OECD* and *Other* [OECD is the Organization for Economic Cooperation and Development, an international think tank charged with promoting policies that will improve global social and economic well-being] then to regress on group we might set

$$x_{i1} = \begin{cases} 1 & \text{if } i\text{th country is in OECD} \\ 0 & \text{if } i\text{th country is not in OECD} \end{cases}$$

and

$$x_{i2} = \begin{cases} 1 & \text{if } i\text{th country is Other} \\ 0 & \text{if } i\text{th country is not Other} \end{cases}$$

Then

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \varepsilon_i = \begin{cases} \beta_0 + \varepsilon_i & \text{if } i\text{th country is in Africa} \\ \beta_0 + \beta_1 + \varepsilon_i & \text{if } i\text{th country is in OECD} \\ \beta_0 + \beta_2 + \varepsilon_i & \text{if } i\text{th country is neither in Africa or OECD} \end{cases}$$

Thus, β_0 is the average life expectancy in Africa, β_1 is the additional expectancy for a person in an OECD country and β_2 is the additional expectancy for a person in a different (non-African or OECD) country. For instance, we found

$$\hat{\beta}_0 = 59.8 \qquad \hat{\beta}_1 = 22.7 \qquad \hat{\beta}_2 = 15.6$$

so that life expectancy in Africa is 59.8, life expectancy in OECD countries is $59.8 + 22.7 = 82.5$ and everywhere else is $59.8 + 15.6 = 75.4$ years. In this case, β_0 is known as the *baseline*. Switching coding will keep predictions the same, but will change interpretations of coefficients.

Bad idea: set

$$x_i = \begin{cases} 0 & \text{if } i\text{th country is Africa} \\ 1 & \text{if } i\text{th country is OECD (switch to Other)} \\ 2 & \text{if } i\text{th country is Other (switch to OECD)} \end{cases}$$

[Picture of how linear relationship turns sour if switch Other/OECD]

Beyond Additivity and Linearity

Two restrictive assumptions we have made so far are *additivity* and *linearity*. The additive assumption implies the effect of one predictor on Y is independent of the values of the other predictors. The linearity assumption implies that the change in Y for a one-unit change in X does *not* depend on the value of X.

The usual generalization to remove the additive assumption is to consider *interactions*. A simple model with an interaction is

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2 + \varepsilon.$$

For instance, if Y is salary, X_1 is years since degree and X_2 is 1 for male (0 for female), then β_1 is the average raise per year for females, while $\beta_1 + \beta_3$ is the average raise per year for males, and β_0 is the average starting salary for females while $\beta_0 + \beta_2$ is the average starting salary for males. The *hierarchical principle* states that if an interaction is to be included, then the *main effects* of X_1 and X_2 alone should also be included, even if their coefficients are not significantly different than zero.

To overcome linearity, *polynomial regression* is usually the first step. For example, to generalize

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

we could consider

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \varepsilon$$

which fits a *quadratic* function to the data, but note the model is still a *linear model* in that it is *linear in the coefficients*. This is equivalent to using two predictors $X_1 = X$ and $X_2 = X^2$. We will spend a lot of time later on more methods for nonlinear modeling. Variable selection techniques can still be used for each.

Transformations

Recall assumptions A1:

1. $Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon$
2. $\mathrm{Var}\varepsilon_i = \sigma^2$ for all i.
3. ε_i are normally distributed
4. ε_i are independent (uncorrelated)

If these are not met, then we can either transform the response, predictors or both. Transformation of Y or X can help ensure 1, whereas only transformations of Y can help ensure 2-3. Data transformation cannot help meeting assumption 4.

We've seen polynomial regression as a possible way to transform covariates, and will focus on response transformations. If $Y > 0$ is a strictly positive then a log transformation $\log Y$ is common (and often helps ensure assumption 2). Other common possibilities are square-root or reciprocal ($\sqrt{Y}$ or $1/Y$). Note that if we use

$$\log Y = \beta_0 + \beta_1 X + \varepsilon$$

then

$$Y = e^{\beta_0} e^{\beta_1 X} e^{\varepsilon},$$

or, in other words, the errors are *multiplicative* on the original scale.

Another common family of transformation is the *Box-Cox* family,

$$g_\lambda(Y) = \begin{cases} \frac{Y^\lambda - 1}{\lambda} & \lambda \neq 0 \\ \log Y & \lambda = 0. \end{cases}$$

where we would then model, e.g.,

$$g_\lambda(Y) = \beta_0 + \beta_1 X + \varepsilon.$$

The best value of λ is usually chosen by maximizing the profile-likelihood.

Degrees of Freedom

If we fit the model

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon$$

we say that we have used $(p+1)$ *degrees of freedom* due to the estimation of the $p+1$ parameters. However, in the future it will be more difficult to define an analogous quantity. If $\mathbf{H}$ is the hat matrix, then note

$$\begin{aligned}\sum_{i=1}^{n} H_{ii} &= \text{tr}(\mathbf{H}) \\ &= \text{tr}(\mathbf{X}(\mathbf{X}^{\text{T}}\mathbf{X})^{-1}\mathbf{X}^{\text{T}}) \\ &= \text{tr}(\mathbf{X}^{\text{T}}\mathbf{X}(\mathbf{X}^{\text{T}}\mathbf{X})^{-1}) \\ &= \text{tr}(\mathbf{I}_{p+1}) = p+1.\end{aligned}$$

Recalling that H_{ii} is the leverage of the ith data point, then $p+1 = \sum_{i=1}^{n} H_{ii}$ is the total influence of all observations, so the greater the number of parameters the greater the aggregate influence of the observations. We will see later that semiparametric regression methods also fall into a class of predictors like $\hat{\mathbf{y}} = \mathbf{M}\mathbf{y}$, where we can define $\text{tr}(\mathbf{M})$ as the *effective degrees of freedom.*

[Audi.R]

5 Classification

Classification refers to the case where the outcome of interest, Y, is *categorical* or *qualitative*, rather than quantitative. For instance, we might want to predict whether a

- New e-mail is spam or not
- Person will default on home mortgage
- Person is likely to have heart disease as they age
- Person will buy a product if shown a (or some) ads for it

based on a set of possibly useful covariates. Classification can be for either *binary* outcomes (spam or not) or *multivariate/multinomial* outcomes (mutations in a stretch of DNA are associated with different phenotypes). Common methods are

- Logistic regression (and probit regression)
- Discriminant analysis
- K-nearest neighbors
- Support vector machines

Note that if $Y \in \{0, 1\}$ we don't want to use regression since typically $\mathbf{x}^{\mathrm{T}}\hat{\boldsymbol{\beta}} \neq 0$ or 1. An alternative is to model the probabilities $P(Y = 0)$ and $P(Y = 1)$. If we can model this probability, then the *classifier* (also known as the *classification rule*) is just a rule that assigns some probabilities to 1 and others to 0 (e.g., predict $Y = 1$ if $P(Y = 1) > 0.5$).

5.1 Logistic Regression

Let $p(X) = P(Y = 1|X)$, where X is a covariate. Why wouldn't we attempt a model like

$$p(X) = \beta_0 + \beta_1 X?$$

Instead, logistic regression is based on the *logistic function*

$$g(x) = \frac{e^x}{1+e^x}$$

[Picture of $e^x/(1+e^x)$, note $x \to -\infty$ and $x \to \infty$] from which we model our probabilities:

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}. \tag{5}$$

Note that $p(X) \in (0,1)$ for all X. There are other functions that transform a regression to $(0,1)$, such as the *probit model*

$$p(X) = \Phi(\beta_0 + \beta_1 X)$$

where Φ is the cdf of a $N(0,1)$.

The *logit transform* is the inverse of the logistic function,

$$f(p) = \log\left(\frac{p}{1-p}\right)$$

and so the assumed logistic regression model (inverting (5)) is equivalent to

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X.$$

The fraction

$$\frac{p(X)}{1-p(X)}$$

is called the *odds ratio*.

Thus, the logistic regression model is linear in the *log odds-ratio*: for every unit increase in X, the log odds-ratio increases by β_1. The odds $p/(1-p)$ take on values in $[0,\infty)$, so, for instance, if $p = 0.5$, then the odds are even, whereas if $p = 0.2$, the odds are $1/4$. Note that the *sign* of β_1 tells us the direction of influence of X on $P(Y = 1|X)$: $\beta_1 > 0$ implies probability grows with X and $\beta_1 < 0$ implies probability is inversely related to X.

In this setup, $Y \sim \text{Bernoulli}(p(X))$. If we assume $Y_1, \ldots, Y_n$ are independent, then given training data $(x_1, y_1), \ldots, (x_n, y_n)$, the likelihood function for $\mathbf{y}$ is

$$f(\mathbf{y}) = \prod_{i=1}^{n} p(x_i)^{y_i}(1-p(x_i))^{1-y_i}.$$

Estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ are found by maximizing $f(\mathbf{y})$, resulting in MLEs. Closed form solutions for $\hat{\beta}_0$ and $\hat{\beta}_1$ don't exist, but they can be approximated numerically.

Sanity check: suppose no covariate, so $p(x) = p = \exp(\beta_0)/(1+\exp(\beta_0))$. Then

$$f(\mathbf{y}) = \prod_{i=1}^{n} p^{y_i}(1-p)^{1-y_i} = p^{n\overline{y}}(1-p)^{n-n\overline{y}}.$$

The derivative of $\log f(\mathbf{y})$ with respect to p is

$$\frac{\mathrm{d}}{\mathrm{d}p}\log f(\mathbf{y}) = \frac{n\overline{y}}{p} - \frac{n-n\overline{y}}{1-p} = 0$$

implies

$$\hat{p} = \overline{y}.$$

Equivalently, $\hat{\beta}_0 = \log(\overline{y}/(1-\overline{y}))$ so that $\hat{\beta}_0$ is just the empirical log odds ratio. In the absence of covariates, logistic regression just uses the proportion of positives as the estimate of p.

Multiple logistic regression follows analogously,

$$P(Y=1|X_1,\dots,X_p) = p(X_1,\dots,X_p) = p(\mathbf{X}) = \frac{e^{\beta_0+\sum_{i=1}^{p}\beta_i X_i}}{1+e^{\beta_0+\sum_{i=1}^{p}\beta_i X_i}},$$

or equivalently

$$\log\left(\frac{p(\mathbf{X})}{1-p(\mathbf{X})}\right) = \beta_0 + \sum_{i=1}^{p}\beta_i X_i.$$

Similar concepts from linear regression are appropriate for logistic regression as well, e.g., the z-statistic is based off of $\hat{\beta}_1/SE(\hat{\beta}_1)$, and can be used to generate p-values or test $H_0 : \beta_1 = 0$.

For a set of new covariates $\mathbf{x} = (x_1,\dots,x_p)^{\mathrm{T}}$, our prediction for $p(\mathbf{x})$ is

$$\hat{p}(\mathbf{x}) = \frac{e^{\hat{\beta}_0+\sum_{i=1}^{p}\hat{\beta}_i x_i}}{1+e^{\hat{\beta}_0+\sum_{i=1}^{p}\hat{\beta}_i x_i}}.$$

[but this gives a probability, not a decision to forecast as 0/1]

The usual classification rule is

$$\hat{y} = \begin{cases} 1 & \hat{p}(\mathbf{x}) > 0.5 \\ 0 & \hat{p}(\mathbf{x}) < 0.5 \end{cases}$$

(with randomization for $\hat{p}(\mathbf{x}) = 0.5$). Note, then

$$\frac{1}{2} < \hat{p} \iff \frac{1}{2} < \frac{e^x}{1+e^x} \iff \frac{1}{2} < \frac{1}{2}e^x \iff 0 < x.$$

Thus, this decision rule is equivalent to

$$\hat{y} = \begin{cases} 1 & \hat{\beta}_0 + \mathbf{x}^{\mathrm{T}}\hat{\boldsymbol{\beta}} > 0 \\ 0 & \hat{\beta}_0 + \mathbf{x}^{\mathrm{T}}\hat{\boldsymbol{\beta}} < 0. \end{cases}$$

This is also known as the *Bayes classifier*. The line $\hat{\beta}_0 + \mathbf{x}^{\mathrm{T}}\hat{\boldsymbol{\beta}} = 0$ is the *decision boundary*, and note that it is *linear* in the covariates. Thus, for prediction/classification, we don't need to calculate probabilities, only the sign of the regression line.

Assessing Quality of Model Fit

Given a set of predictions, $\hat{y}_1, \ldots, \hat{y}_n \in \{0, 1\}$, how do we assess quality of fit? In this section think of 1 ="an email is spam" and 0 ="an email is not spam".

We can calculate the *error rate*

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{[y_i \neq \hat{y}_i]} = \text{percent misspecified}$$

and look at a *confusion matrix*

	True # of 1s	True # of 0s
Predicted # of 1s	10	4
Predicted # of 0s	2	12

From here, we can read off the

- *Sensitivity*: percent of true positives (10/12)
- *Specificity*: percent of true negatives (12/16)
- *Positive predictive value*: % of 1s correctly identified (10/14)
- *Negative predictive value*: % of 0s correctly identified (12/14)
- *Error rate*: $(4 + 2)/28$.

We need specificity, e.g., since if we only cared about predicting positives, we would always use 1 as the classifier, regardless of covariate. Additionally define

- *False positive rate (FPR)* = 4/16 = 1 - specificity
- *True positive rate (TPR)* = 10/12 = sensitivity

Goal: try to *minimize* FPR and simultaneously *maximize* TPR. Here, FPR quantifies "good emails going to the spam folder" while TPR quantifies "spam emails going to the spam folder."

These should be compared against random guessing. For example: if the data had 100 positives and 1000 negatives, what might the confusion matrix look like for guessing 1 with 90% probability every time? [Go to table] Thus, we would get $TPR = 0.9$ and $FPR = 0.9$. To improve on this, for *any fixed FPR*, we want to have *higher TPR*.

	True # of 1s	True # of 0s
Predicted # of 1s	90	900
Predicted # of 0s	10	100

A *ROC (receiver operating characteristic) graph*, plots FPR (1-specificity) on the x-axis and TPR (sensitivity) on the y-axis. The point at $(0, 1)$ indicates perfect predictions, and the diagonal line indicates random guessing (thus curves closer to the upper left corner indicate better models).

[Picture]

For any given model and decision rule, we get a single *point* in ROC space. But what if we took our classification threshold ($p(X) > 0.5$ implies classify as 1), and tried to vary the threshold 0.5? For each different threshold we consider, we would get a *different* confusion matrix. By considering *many* thresholds from $[0, 1]$ we can create a *ROC curve*, which can be compared against the identity line, and also against competing classification procedures.

Thought experiment: What would the TPR and FPR be for a threshold of 0 (that is, always predict positive)? What would the TPR/FDR be for a threshold of 1 (always predictive negative)? What would the best point on a ROC plot be?

[Upper right-hand corner classifiers are more liberal – they make positive classifications with weak evidence, but their FPR is high. Lower left-hand corner classifiers are conservative – fewer positive classifications leads to fewer false positives]

A numerical summary of *area under ROC (AUC)* is often used, and is compared to 0.5, which is the expected AUC under chance guessing.

In linear regression, RSS is used as a measure of model fit. The analogous quantity for logistic regression is the *deviance*, denoted G^2 where

$$G^2 = 2\sum_{i=1}^{n}\left[y_i \log\left(\frac{y_i}{\hat{p}(\mathbf{x}_i)}\right) + (1-y_i)\log\left(\frac{1-y_i}{1-\hat{p}(\mathbf{x}_i)}\right)\right].$$

G^2 can be used to test for differences between models by comparing against an appropriate χ^2 distribution, with smaller values indicating better fit. We define the *deviance residual* as

$$dev_i = \pm\left(-2\big[y_i\log(\hat{p}(\mathbf{x}_i)) + (1-y_i)\log(1-\hat{p}(\mathbf{x}_i))\big]\right)^{1/2}$$

where $\pm$ is positive if $y_i \geq \hat{p}(\mathbf{x}_i)$. dev_i is the (signed square root) contribution of the ith data point to the total deviance. [Thought experiment: what does dev_i look like for $y_i = 1$ and $\hat{p}(\mathbf{x}_i)$ close to 1 or 0?]

Without the decision rule, logistic regression predictions are *probabilities.* Require some way of *quantifying quality of a probabilistic forecast.* These are called *scoring rules.* Given a set of predicted probabilities $\hat{p}(\mathbf{x}_1), \ldots, \hat{p}(\mathbf{x}_n)$ and observations $y_1, \ldots, y_n \in \{0, 1\}$, the *Brier score* is

$$BS = \frac{1}{n}\sum_{i=1}^{n}(\hat{p}(\mathbf{x}_i) - y_i)^2.$$

Lower Brier scores indicate better performance. [Note where minimum occurs]

For classification problems, it is *crucial* to split the data into *training* and *testing* data. It turns out that the *in-sample* error (that is, the error on the data used to train the model) is almost always better than the *out-of-sample* error (testing data).

[R example (LogisticRegression.R)]

5.2 Discriminant Analysis

Now suppose $Y \in \{1, \ldots, K\}$ falls into one of K *classes*. Discriminant analysis takes a slightly different approach by assuming a distribution on the predictor $[X|Y]$ and inverting via Bayes' formula to predict $[Y|X]$. Some notation:

- $\pi_k = P(Y = k)$ is the *prior probability* that Y falls in the kth class
- $f_k(x) = P(X = x|Y = k)$ is the conditional density function for X given that Y is in class k

Then we have

$$\begin{aligned} p_k(x) = P(Y = k|X = x) &= \frac{P(X = x|Y = k)P(Y = k)}{P(X = x)} \\ &= \frac{f_k(x)\pi_k}{\sum_{i=1}^{K} f_i(x)\pi_i} \end{aligned}$$

defines the *posterior probability* that Y will be in the kth class given $X = x$.

For a new feature $X = x$, the usual decision rule is to classify Y as the class k for which $p_1(x), \ldots, p_K(x)$ is maximized. For a fixed x, the denominator is constant across $p_k(x)$, so finding the *most probable class* is equivalent to finding which of $\pi_1 f_1(x), \ldots, \pi_K f_K(x)$ is greatest.

Linear Discriminant Analysis for $p = 1$

Linear discriminant analysis (LDA) *assumes* $[X|Y = k]$ are approximately normally distributed with all conditionals having the same variance, so

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma}\right)^2}.$$

Classification of $[Y|X = x]$ boils down to finding the maximal (over k)

$$\begin{aligned} \log(\pi_k f_k(x)) &= C_1 + \log \pi_k - \frac{1}{2\sigma^2}(x^2 - 2x\mu_k + \mu_k^2) \\ &= C_1 + \log \pi_k + x\frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + C_2(x) \\ &= \log \pi_k + x\frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \text{constants that don't depend on } k. \end{aligned}$$

Thus, finding the k for which the *discriminant function*

$$\delta_k(x) = \log \pi_k + x\frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2}$$

is maximized is the classification rule for linear discriminant analysis (note that it is linear in x, hence the name). In the special case of two classes ($K = 2$) and $\pi_1 = \pi_2$, we assign $[Y|X = x]$ to class 1 if

$$\log \pi_1 + x\frac{\mu_1}{\sigma^2} - \frac{\mu_1^2}{2\sigma^2} > \log \pi_2 + x\frac{\mu_2}{\sigma^2} - \frac{\mu_2^2}{2\sigma^2}$$

$$\iff$$

$$x\mu_1 - \mu_1^2/2 > x\mu_2 - \mu_2^2/2$$

$$2x(\mu_1 - \mu_2) > \mu_1^2 - \mu_2^2,$$

and to class 2 otherwise. The *decision boundary* is

$$2X(\mu_1 - \mu_2) = \mu_1^2 - \mu_2^2 \iff X = \frac{\mu_1^2 - \mu_2^2}{2(\mu_1 - \mu_2)} = \frac{\mu_1 + \mu_2}{2}.$$

Example 2. *Let $Y = \{0, 1\}$ for being not spam/spam e-mail, and let X denote log of length of longest string of capital letters. Then $\pi_0 \approx 0.6$ and $\pi_1 \approx 0.4$ in our study dataset.*

For a new e-mail with $X = x$, the decision rule is

$$\hat{Y} = \begin{cases} 1 & \delta_1(x) > \delta_0(x) \\ 0 & \delta_1(x) < \delta_0(x). \end{cases}$$

[Picture of mock f_0 and f_1 with about the same variance, $\mu_1 > \mu_0$, with two example new e-mails: 1) in right tail of f_1, 2) exactly at $f_0(x) = f_1(x)$ so that the decision reverts to the population rate comparison].

Estimation of μ_k, π_k and σ^2 follow by

- $\hat{\pi}_k = \frac{n_k}{n}$
- $\hat{\mu}_k = \frac{1}{n_k} \sum_{i|y_i=k} x_i$
- $\hat{\sigma}^2 = \frac{1}{n-K} \sum_{k=1}^{K} \sum_{i|y_k=k} (x_i - \hat{\mu}_k)^2$

where n_k is the total number of observations in class k and $n = \sum_{i=1}^{K} n_k$ is the total number of observations.

Linear Discriminant Analysis for $p > 1$

If there are $\mathbf{X} = (X_1, \dots, X_p)^{\mathrm{T}}$ covariates per observation, then the extension of LDA is that

$$[\mathbf{X}|Y = k] \sim N_p(\boldsymbol{\mu}_k, \Sigma)$$

where $\boldsymbol{\mu}_k = (\mathbb{E}X_1, \dots, \mathbb{E}X_p)^{\mathrm{T}}$ and $\Sigma = \{\mathrm{Cov}(X_i, X_j)\}_{i,j=1}^{p}$. The Bayes classifier in this case for an observation $\mathbf{X} = \mathbf{x}$ is the class k for which

$$\delta_k(\mathbf{x}) = \mathbf{x}\Sigma^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}_k^{\mathrm{T}}\Sigma^{-1}\boldsymbol{\mu}_k + \log \pi_k$$

is maximized. Note it is still linear in $\mathbf{x}$. Estimation is analogous to $p = 1$, for observation pairs $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$, let

- $\hat{\pi}_k = \frac{n_k}{n}$
- $\hat{\boldsymbol{\mu}}_k = \frac{1}{n_k}\sum_{i|y_i=k}\mathbf{x}_i$
- $\hat{\Sigma} = \frac{1}{n-K}\sum_{k=1}^{K}\sum_{i|y_i=k}(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_k)(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_k)^{\mathrm{T}}$.

Quadratic Discriminant Analysis

Quadratic discriminant analysis (QDA) relaxes the assumption that Σ is constant across classes $k = 1, \dots, K$. That is, in QDA we assume $[\mathbf{X}|Y = k] \sim N_p(\boldsymbol{\mu}_k, \Sigma_k)$. The Bayes classifier for Y given $\mathbf{X} = \mathbf{x}$ is the class k that maximizes

$$\delta_k(\mathbf{x}) = -\frac{1}{2}\mathbf{x}^{\mathrm{T}}\Sigma_k^{-1}\mathbf{x} + \mathbf{x}^{\mathrm{T}}\Sigma_k^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}_k^{\mathrm{T}}\Sigma_k^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\log|\Sigma_k| + \log \pi_k.$$

Note $\delta_k(\mathbf{x})$ is *quadratic* in $\mathbf{x}$.

QDA seems more general than LDA, when would we opt for one or the other?

LDA: Large p or n not much bigger than p

QDA: Small p or $n \gg p$.

If we have lots of covariates, then QDA will attempt to estimate p covariance matrices Σ_k where the number of parameters scales like $p(p+1)/2$ – we will incur much variability for lower bias. In these cases, LDA will reduce variability at the cost of some bias, which may be desirable.

Warning: LDA and QDA are *not* appropriate for categorical covariates. [Thought experiment: can you think of a generalization to allow for categorical covariates?]

[R example (DiscriminantAnalysis.R)]

5.3 K-nearest Neighbors

K-nearest neighbors (KNN) is a simple, nonparametric method for classification which estimates probabilities $P(Y = j|X = x)$ by empirically averaging nearby observations to x. In particular, the KNN estimate is

$$\hat{P}(Y = j|X = x) = \frac{1}{K} \sum_{i \in \mathcal{N}_x} \mathbb{1}_{[y_i = j]}$$

where K is some pre-specified integer and $\mathcal{N}_x$ is the set of K nearest training points to x.

[Picture]

Note that if $K = 1$ or K is small, the decision boundary will typically be rather erratic (high variance, but low bias), whereas if K is large then there will be reduced variance (but high bias).

It is particularly important to split the data into *training* and *testing* sets. In general a chosen method will perform better on a training dataset than the testing data. For instance, if we use the whole dataset to train, then the estimated error rate for KNN with $K = 1$ will be zero, but will surely suffer with the introduction of new points.

[R example (KNN.R)]

6 Regularization

Recall the basic multiple linear regression problem. We model

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon = \mathbf{x}^{\mathrm{T}}\boldsymbol{\beta} + \varepsilon.$$

If we have many predictors, $p \gg 0$, then our usual OLS estimator

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

will exhibit high variances and will be unstable. Worse, if we have $p > n$ predictors, then the OLS estimators are unidentifiable, since there are infinitely many solutions to

$$(\mathbf{X}^{\mathrm{T}}\mathbf{X})\boldsymbol{\beta} = \mathbf{X}^{\mathrm{T}}\mathbf{y}.$$

Rather than resorting to a subset selection method, the goal of this section is to include *all* variables, but to *penalize* or *regularize* their coefficients so as to encourage them to be closer to zero. We will see that this provides a reduction in the *effective degrees of freedom*.

Given samples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ where $\mathbf{x} = (1, x_1, \ldots, x_p)$, the usual OLS estimator minimizes

$$\sum_{i=1}^{n}(y_i - \mathbf{x}_i^{\mathrm{T}}\boldsymbol{\beta})^2 = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}).$$

Regularization follows by adding a *penalty* term to this,

$$(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) \to (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda P(\boldsymbol{\beta})$$

where $P(\boldsymbol{\beta})$ is a penalty that increases as $\|\boldsymbol{\beta}\| \to \infty$, and λ is known as a *complexity parameter*, *smoothing parameter* or *shrinkage parameter* that controls the amount of weight put on the penalty. Thus, to minimize this over $\boldsymbol{\beta}$, we must *balance* between model fit and "model size." Why does $\|\boldsymbol{\beta}\|$ control "model size"? Well if $\|\boldsymbol{\beta}\| = 0$ then $\boldsymbol{\beta} = \mathbf{0}$ and the model is that Y does not depend on any X_i, which is a very small model, whereas if $\|\boldsymbol{\beta}\| \to \infty$ where each argument tends to ∞ then the model is that Y depends *strongly* on all features, i.e., the most complex model we entertain.

6.1 Ridge Regression

β_0 only measures the average value of Y, and should not be penalized.

Note the following useful fact when using *centered* covariates:

$$\begin{aligned}\frac{\partial}{\partial \beta_0}\sum_{i=1}^n (y_i - \beta_0 - \beta_1(x_i - \overline{x}))^2 &= -2\sum_i (y_i - \beta_0) + 2\beta_1 \sum_i (x_i - \overline{x}) \\ &= -2n\overline{y} + 2n\beta_0\end{aligned}$$

which implies the OLS estimator for β_0 is $\hat{\beta}_0 = \overline{y}$. In this section we will always work with a centered response and centered/scaled features.

The basic idea behind *ridge regression* is to penalize the *squared* L_2 *length* of the coefficient vector $\boldsymbol{\beta} = (\beta_1, \dots, \beta_p)^{\mathrm{T}}$. That is, using a penalty of the form $P(\boldsymbol{\beta}) = \sum_{i=1}^n \beta_i^2 = \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{\beta} = \|\boldsymbol{\beta}\|_2^2$.

Problem: what if $p = 2$ and X_1 is budget of movie and X_2 is rating. Then $X_1 \sim$ 10000s and \$s and $X_2 \sim$ 1s without units. What units is $\|\boldsymbol{\beta}\|_2^2$ in? Does the length of the vector make sense in this case? We need to remove the *dimension* of the β_is!

Thus, for the remainder of this section we will assume

- Observations y_i have been centered by $\overline{y}$, that is, $y_i \to y_i - \overline{y}$.
- Covariates have been centered and scaled, that is,

$$x_i \to \frac{x_i - \overline{x}}{\sqrt{\frac{1}{n}\sum_{i=1}^n (x_i - \overline{x})^2}}$$

Thus, Y is mean zero (at $(X_1, \dots, X_p) = (0, \dots, 0)$) and X_i are *unitless*. Additionally, (with a bit of abuse of notation)

$$\begin{aligned}\sum_{i=1}^n x_i^2 &= \sum_{i=1}^n \left(\frac{x_i - \overline{x}}{\sqrt{\frac{1}{n}\sum_{i=1}^n (x_i - \overline{x})^2}}\right)^2 \\ &= \frac{\sum_{i=1}^n (x_i - \overline{x})^2}{\frac{1}{n}\sum_{i=1}^n (x_i - \overline{x})^2} \\ &= n.\end{aligned}$$

The *ridge regression* solution minimizes

$$\begin{aligned}\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda\sum_{i=1}^{j}\beta_j^2 &= \sum_{i=1}^{n}(y_i - \mathbf{x}_i^{\mathrm{T}}\boldsymbol{\beta})^2 + \lambda\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{\beta} \\ &= (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda\|\boldsymbol{\beta}\|_2^2.\end{aligned}$$

where $\mathbf{x}_i = (x_{i1}, \ldots, x_{ip})^{\mathrm{T}}$ and $\mathbf{X}$ is the usual design matrix but *without the first column of* 1*s*.

Example 3. *Suppose* $p = 1$. *Then*

$$\min_{\beta_1}\sum_{i=1}^{n}(y_i - \beta_1 x_i)^2$$

is

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n x_i y_i}{\sum_{i=1}^n x_i^2} = \frac{\sum_{i=1}^n x_i y_i}{n}.$$

by Homework 1. The ridge solution would set equal to zero (by taking a derivative wrt β_1*)*

$$-2\sum_{i=1}^{n} x_i(y_i - \beta_1 x_i) + 2\lambda\beta_1 = -2\sum_{i=1}^{n} x_i y_i + \beta_1(n + \lambda)$$

implying

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n x_i y_i}{n + \lambda}.$$

Thus, λ shrinks β_1 *toward zero!*

Ridge regression is sometimes known as a *shrinkage estimator*.

If $\lambda = 0$, then ridge regression reduces to OLS. If $\lambda \to \infty$, then the minimization pushes $\boldsymbol{\beta} \to \mathbf{0}$, and the model reduces to $Y = \beta_0 + \varepsilon$.

Let's derive the ridge solution:

$$\begin{aligned}\frac{\partial}{\partial\boldsymbol{\beta}}\left[(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{\beta}\right] &= -2\mathbf{X}^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + 2\lambda\boldsymbol{\beta} \\ &= 2\left((\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{I})\boldsymbol{\beta} - \mathbf{X}^{\mathrm{T}}\mathbf{y}\right)\end{aligned}$$

which implies

$$\hat{\boldsymbol{\beta}}_{ridge} = (\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}.$$

Note that $(\mathbf{X}^{\mathrm{T}}\mathbf{X})$ is a nonnegative definite matrix [show]; thus, it has eigenvalues that are nonnegative. However, if $(\mathbf{X}^{\mathrm{T}}\mathbf{X})$ is not of full rank, p, it has some zero eigenvalues and is not invertible. The effect of $+\lambda\mathbf{I}$ is to bump these zero eigenvalues to λ, and $\mathbf{X}^{\mathrm{T}}\mathbf{X}+\lambda\mathbf{I}$ is invertible. Thus, ridge regression allows us to fit models with $p > n$ covariates.

What is ridge regression doing? Note that $\hat{\boldsymbol{\beta}}_{ridge}$ is biased,

$$\mathbb{E}\hat{\boldsymbol{\beta}}_{ridge} = (\mathbf{X}^{\mathrm{T}}\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{X}\boldsymbol{\beta} \neq \boldsymbol{\beta}.$$

On the other hand,

$$\mathrm{Var}\hat{\boldsymbol{\beta}}_{ridge} = \sigma^2(\mathbf{X}^{\mathrm{T}}\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{X}(\mathbf{X}^{\mathrm{T}}\mathbf{X}+\lambda\mathbf{I})^{-1},$$

which should be compared against $\mathrm{Var}\hat{\boldsymbol{\beta}}_{OLS} = \sigma^2(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}$. Thus ridge regression is *sacrificing some bias to reduce the variability* in the estimates.

Example 4. *In the previous example, we had the OLS estimator for*

$$y_i = \beta_1 x_i + \varepsilon_i$$

is unbiased:

$$\mathbb{E}\left(\frac{\sum_{i=1}^n x_i y_i}{\sum_{i=1}^n x_i^2}\right) = \frac{\beta_1\sum_{i=1}^n x_i^2}{\sum_{i=1}^n x_i^2} = \beta_1.$$

The OLS estimator's variance is

$$\mathrm{Var}\left(\frac{\sum_{i=1}^n x_i y_i}{\sum_{i=1}^n x_i^2}\right) = \frac{\sigma^2\sum_{i=1}^n x_i^2}{\left(\sum_{i=1}^n x_i^2\right)^2} = \frac{\sigma^2}{n}.$$

The ridge expectation is

$$\mathbb{E}\left(\frac{\sum_{i=1}^n x_i y_i}{n+\lambda}\right) = \frac{n}{n+\lambda}\beta_1,$$

and so is a biased *estimator. However,*

$$\mathrm{Var}\left(\frac{\sum_{i=1}^n x_i y_i}{n+\lambda}\right) = \frac{n}{(n+\lambda)^2}\sigma^2,$$

and so reduces the variance over the OLS estimator.

Why would we want to sacrifice some bias in order to reduce the variability? Recall that the *mean squared error* is defined as

$$MSE = \mathbb{E}\sum_{i=1}^p\left(\beta_i - \hat{\beta}_i\right)^2 = \mathbb{E}(\boldsymbol{\beta}-\hat{\boldsymbol{\beta}})^{\mathrm{T}}(\boldsymbol{\beta}-\hat{\boldsymbol{\beta}}).$$

For ridge regression it's not too difficult to show that

$$MSE(\lambda) = \boldsymbol{\beta}^{\mathrm{T}}(\mathbf{I} - \mathbf{M}(\lambda)\mathbf{X})^{\mathrm{T}}(\mathbf{I} - \mathbf{M}(\lambda)\mathbf{X})\boldsymbol{\beta} + \sigma^2 \mathrm{tr}(\mathbf{M}(\lambda)^{\mathrm{T}}\mathbf{M}(\lambda))$$

where $\mathbf{M}(\lambda) = (\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^{\mathrm{T}}$. The first term can be interpreted as the squared bias of the ridge estimator, while the second is the sum of variances of the estimators. For certain choices of λ, ridge regression has lower MSE than ordinary least squares, which is why sacrificing some bias for lower variance may be useful.

How do we choose λ? The hat matrix for ridge regression is

$$\mathbf{H}(\lambda) = \mathbf{X}(\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^{\mathrm{T}}$$

and is independent of $\mathbf{y}$, but depends on λ. If y_k is the kth data point, $\hat{y}_k$ is the fitted value based on all n data points and $\hat{y}_{-k}$ is the fitted value based on the $(n-1)$ data points with y_k removed, then call $y_k - \hat{y}_{-k}$ the cross-validated residual. We then have

$$y_k - \hat{y}_{-k} = \frac{y_k - \hat{y}_k}{1 - \mathbf{H}(\lambda)_{kk}}.$$

We could choose lambda to minimize the leave-one-out cross-validated residual sum of squares,

$$\sum_{k=1}^{n}(y_k - \hat{y}_{-k})^2 = \sum_{k=1}^{n}\left(\frac{y - \hat{y}_k}{1 - \mathbf{H}(\lambda)_{kk}}\right)^2.$$

Note a nice thing about this identity is that we don't have to *recalculate* $\hat{\boldsymbol{\beta}}$ each time we hold out a particular k.

An alternative is to use generalized cross-validation (GCV), where we minimize

$$\sum_{k=1}^{n}\left(\frac{y - \hat{y}_k}{1 - \mathrm{tr}\mathbf{H}(\lambda)/n}\right)^2$$

where we are using the *average* of the diagonal of the hat matrix, i.e., the average leverage.

The difference between (leave-one-out) cross-validation and GCV is that in leave-one-out, each estimated residual is scaled by one minus its leverage, whereas we use the total leverage for GCV. Note that if the leverage is very large and close to 1 (for a particular choice of λ), then the cross-validation sum of squares is large. In other words, the optimal λ will not allow any one (or few) data points to have large leverage over the final fit, and seeks to spread out the total leverage over data points, while still maintaining reasonable predictions.

[R example (RidgeRegression.R)]

6.2 Lasso

One drawback to ridge regression is that the estimated β_is almost always take on positive values, even if negligibly small. To overcome this, the *lasso* (least absolute shrinkage and selection operator) is an alternative that minimizes

$$(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})^{\mathrm{T}}(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})+\lambda\|\boldsymbol{\beta}\|_1=(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})^{\mathrm{T}}(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})+\lambda\sum_{i=1}^{p}|\beta_i|$$

where the $\|\boldsymbol{\beta}\|_2^2$ has been replaced by the L_1 norm.

Lasso has the same effect of *shrinking* the coefficients closer to zero, but, amazingly, the lasso solution tends to threshold some covariates to *exactly* zero, and thus also acts as a variable selection tool. In other words, the lasso generates *sparse* models, that is, models with only a subset of variables.

6.3 Another View of Ridge and Lasso: Lasso as a Variable Selector

Note an equivalent way to write the ridge problem is as

$\min_{\boldsymbol{\beta}}\sum_{i=1}^{n}(y_i-\boldsymbol{\beta}^{\mathrm{T}}\mathbf{x}_i)^2$

subject to $\sum_{j=1}^{p}\beta_j^2 \le s$.

while lasso is equivalent to

$\min_{\boldsymbol{\beta}}\sum_{i=1}^{n}(y_i-\boldsymbol{\beta}^{\mathrm{T}}\mathbf{x}_i)^2$

subject to $\sum_{j=1}^{p}|\beta_j| \le s$.

We are minimizing the squared error (as in OLS), *subject to* living in either a L_2 (ridge) or L_1 (lasso) sphere of radius s. Note that if $s=0$, our model reduces to $Y=\beta_0+\varepsilon$, while as $s\to\infty$, we return the OLS estimators.

[Picture: try to find contours closest to OLS estimator that intersect the L_1/L_2 spheres, Figure 6.7 ISLR]

[R example (Lasso.R)]

6.4 A Bayesian View of Ridge and Lasso

In Bayesian statistics, we represent initial beliefs about a parameter β using a *prior* distribution and, given data, update these beliefs by formulating a *posterior* distribution.

If $\mathbf{y}$ are observations from $Y = \beta X + \varepsilon$ under A1, then $\mathbf{y}$'s pdf is

$$f(\mathbf{y}|\beta) = \frac{1}{(2\pi\sigma^2)^{n/2}} e^{-\frac{1}{2\sigma^2}(\mathbf{y}-\beta\mathbf{x})^{\mathrm{T}}(\mathbf{y}-\beta\mathbf{x})}.$$

Note that $f(\mathbf{y})$ is bad notation, since it depends on the *unknown* β. Then, given a *prior distribution* $\pi(\beta)$, the goal is to find the *posterior distribution*,

$$f(\beta|\mathbf{y}) = \frac{f(\mathbf{y}|\beta)\pi(\beta)}{f(\mathbf{y})}.$$

$\pi(\beta)$ represents our beliefs about β before any data is available, while $f(\beta|\mathbf{y})$ represents our *updated beliefs about β given the observation* $\mathbf{y}$.

[Picture]

The β with *highest probability* given the data y, is the *posterior mode*, i.e., it is the value of β that maximizes $f(\beta|\mathbf{y})$. Maximizing $f(\beta|\mathbf{y})$ is equivalent to maximizing $\log f(\beta|\mathbf{y})$ is equivalent to minimizing $-\log f(\beta|\mathbf{y})$. Thus, the posterior mode *minimizes*

$$\begin{aligned} -\log(f(\beta|\mathbf{y})) &= -\log(f(\mathbf{y}|\beta)) - \log(\pi(\beta)) + C \\ &= \frac{1}{2\sigma^2}(\mathbf{y}-\beta\mathbf{x})^{\mathrm{T}}(\mathbf{y}-\beta\mathbf{x}) - \log(\pi(\beta)) + C \end{aligned}$$

where C does not depend on β. The ridge solution is the minimizer of

$$(\mathbf{y}-\beta\mathbf{x})^{\mathrm{T}}(\mathbf{y}-\beta\mathbf{x}) + \lambda\beta^2,$$

so set

$$-\log(\pi(\beta)) = \lambda\beta^2$$

and note

$$\pi(\beta) \propto e^{-\lambda\beta^2}.$$

In other words, if we use a normal, mean zero prior for β then *the ridge solution is the same as the posterior mode.*

The lasso solution is the minimizer of

$$(\mathbf{y} - \beta\mathbf{x})^{\mathrm{T}}(\mathbf{y} - \beta\mathbf{x}) + \lambda|\beta|,$$

so set

$$-\log(\pi(\beta)) = \lambda|\beta|$$

and note

$$\pi(\beta) \propto e^{-\lambda|\beta|}.$$

In other words, if we use a double exponential prior (also known as a Laplace distribution) for β then *the lasso solution is the same as the posterior mode.*

7 Classification 2

In this section we'll introduce the *support vector machine*, which generalizes the *support vector classifier* which generalizes the *maximal margin classifier*.

7.1 Maximal Margin Classifier

In p-dimensional space, a *hyperplane* is a flat subspace of dimension $p-1$ – so in 2 dimensions a hyperplane is a line. In 3 dimensions, a hyperplane is just a plane.

Definition 5. *In p-dimensions, a* hyperplane *is all* $\mathbf{x} = (x_1, \ldots, x_p) \in \mathbb{R}^p$ *satisfying*

$$\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p = 0. \tag{6}$$

Note in $p = 2$ this reduces to the equation for a line

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0.$$

[Picture]

If $\mathbf{x}$ does not satisfy (6), then necessarily

$$\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p > 0 \quad \text{or}$$
$$\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p < 0,$$

so the hyperplane divides p-dimensional space into halves.

Suppose we have a set of training data $y_1, \ldots, y_n$ and corresponding covariates $\mathbf{x}_1, \ldots, \mathbf{x}_n$ where

$$y_i \in \{-1, 1\} \quad \mathbf{x}_i = (x_{i1}, \ldots, x_{ip})^{\mathrm{T}}$$

where -1 is one class and 1 is the other.

The goal of a *separating hyperplane* is to find a hyperplane (defined by $\beta_0, \ldots, \beta_p$) such that

$$\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} > 0 \qquad \text{if} \quad y_i = 1$$
$$\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} < 0 \qquad \text{if} \quad y_i = -1$$

This is equivalent to

$$y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) > 0$$

for all $i = 1, \ldots, n$.

[Picture]

Given a test covariate $\mathbf{x}_* = (x_{*1}, \ldots, x_{*p})^{\mathrm{T}}$, the classifier depends on

$$f(\mathbf{x}_*) = \beta_0 + \beta_1 x_{*1} + \cdots + \beta_p x_{*p}$$

$$\hat{y} = \begin{cases} 1 & f(\mathbf{x}_*) > 0 \\ -1 & f(\mathbf{x}_*) < 0 \end{cases}$$

with randomization on the boundary. The magnitude $f(\mathbf{x}_*)$ can be interpreted as a measure of confidence in the decision rule. Note this results in a linear decision boundary.

Problem: there are *many* possible separating hyperplanes, and the choice of any two valid ones will lead to different classification rules.

[Picture]

Separable Case

Suppose the data are *linearly separable* [picture]. The *maximum margin hyperplane* (*optimal separating hyperplane*) is that which is furthest from the training data. This hyperplane defines the *maximal margin classifier* as being above or below the line.

The maximum margin hyperplane is calculated via

maximize$_{\beta_0,\beta_1,\ldots,\beta_p} M$

subject to $\sum_{j=1}^{p} \beta_j^2 = 1$

and $y_i(\beta_0 + \boldsymbol{\beta}^{\mathrm{T}}\mathbf{x}_i) \geq M$ for all $i = 1, \ldots, n$.

Note that restricting $\boldsymbol{\beta}$ to be length one is not restrictive since

$$\beta_0 + \boldsymbol{\beta}^{\mathrm{T}}\mathbf{x} = 0$$

and

$$k(\beta_0 + \boldsymbol{\beta}^{\mathrm{T}}\mathbf{x}) = 0$$

define equivalent hyperplanes for any $k \neq 0$. Additionally,

$$y_i(\beta_0 + \boldsymbol{\beta}^{\mathrm{T}}\mathbf{x}_i)$$

is the distance from the ith observation to the hyperplane, so M is the *margin of the hyperplane*, i.e., the least distance from the data to the hyperplane. To see this, note that $\boldsymbol{\beta}$ is the vector that is normal to the hyperplane. Any vector pointing in the direction of the hyperplane is defined by the difference between two points on the plane – if $\boldsymbol{\ell}_1$ and $\boldsymbol{\ell}_2$ are on the hyperplane then

$$\boldsymbol{\beta}^{\mathrm{T}}(\boldsymbol{\ell}_1 - \boldsymbol{\ell}_2) = \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{\ell}_1 - \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{\ell}_2 = -\beta_0 + \beta_0 = 0.$$

Due to our algorithm, also note $\|\boldsymbol{\beta}\| = 1$, so $\boldsymbol{\beta}$ is a unit vector. Thus the vector from a candidate point $\mathbf{x}_i$ to the line can be written

$$\mathbf{x}_i - \boldsymbol{\ell} = d\boldsymbol{\beta}$$

where d is the signed distance from $\mathbf{x}_i$ to the line. Multiplying by $\boldsymbol{\beta}$ gives

$$\begin{aligned}\boldsymbol{\beta}^{\mathrm{T}}\mathbf{x}_i - \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{\ell} &= d\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{\beta}\\ (\boldsymbol{\beta}^{\mathrm{T}}\mathbf{x}_i + \beta_0) &= d.\end{aligned}$$

Thus, multiplying by y_i gives the unsigned distance.

Defining the Support Vectors

The vectors that are of length M from the hyperplane are known as the *support vectors*, and *only these vectors define the hyperplane* (that is, all other data points are not needed to define the hyperplane).

[Picture]

Recall the original optimization problem

$$\text{maximize}_{\beta_0,\boldsymbol{\beta}} M$$

subject to $\|\boldsymbol{\beta}\|_2 = 1$

and $y_i(\beta_0 + \boldsymbol{\beta}^{\mathrm{T}}\mathbf{x}_i) \geq M$ for all $i = 1, \ldots, n$.

Let's try to remove the $\|\boldsymbol{\beta}\|_2 = 1$ criterion. If $\boldsymbol{\beta}_d$ is a vector parallel to $\boldsymbol{\beta}$ where $\boldsymbol{\beta}_d = d\boldsymbol{\beta}$ (so that $d = \|\boldsymbol{\beta}_d\|_2$) we have

$$\begin{aligned} y_i(\beta_0 + \boldsymbol{\beta}^{\mathrm{T}}\mathbf{x}_i) &\geq M \\ y_i(\beta_0 + \boldsymbol{\beta}_d^{\mathrm{T}}\mathbf{x}_i/d) &\geq M \\ \frac{1}{d}y_i(d\beta_0 + \boldsymbol{\beta}_d^{\mathrm{T}}\mathbf{x}_i) &\geq M \\ y_i(d\beta_0 + \boldsymbol{\beta}_d^{\mathrm{T}}\mathbf{x}_i) &\geq dM. \end{aligned}$$

In other words, the plane defined by $\boldsymbol{\beta}_d$ and $\gamma_0 = d\beta_0$ also satisfies the inequality condition, with bound dM. Thus, setting $d = 1/M$ (which gives a particular γ_0), we have the optimization program can be written

$\min_{\gamma_0, \boldsymbol{\beta}_d} \frac{1}{2}\|\boldsymbol{\beta}_d\|_2^2$

subject to $y_i(\gamma_0 + \boldsymbol{\beta}_d^{\mathrm{T}}\mathbf{x}_i) \geq 1$

because maximizing M in the original procedure is equivalent to minimizing $d = \|\boldsymbol{\beta}_d\|_2$. Now switch notation back to $\beta_0, \boldsymbol{\beta}$.

This can be converted to a Lagrangian (using some advanced optimization theory involving the Karush-Kuhn-Tucker (KKT) conditions),

$$\frac{1}{2}\|\boldsymbol{\beta}\|_2^2 - \sum_{i=1}^{n} \alpha_i(y_i(\beta_0 + \boldsymbol{\beta}^{\mathrm{T}}\mathbf{x}_i) - 1)$$

which we want to minimize over $\beta_0, \boldsymbol{\beta}$ (the α_i's are Lagrange multipliers). Take derivatives wrt $\boldsymbol{\beta}$ and β_0:

- $\boldsymbol{\beta} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$
- $\sum_{i=1}^{n} \alpha_i y_i = 0.$

Substituting back in to the Lagrangian results in the Wolfe dual objective function

$$\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j$$

which we want to maximize subject to $\alpha_i \geq 0$ (it gives a lower bound on the objective for any feasible point). At the solution, either

- $\alpha_i = 0$ which happens if $y_i(\beta_0 + \boldsymbol{\beta}^{\mathrm{T}} \mathbf{x}_i) > 1$ or
- $\alpha_i > 0$ which happens if $y_i(\beta_0 + \boldsymbol{\beta}^{\mathrm{T}} \mathbf{x}_i) = 1$.

If $y_i(\beta_0 + \boldsymbol{\beta}^{\mathrm{T}} \mathbf{x}_i) = 1$, then $\mathbf{x}_i$ is *on* the boundary of the classification margin.

Using $\boldsymbol{\beta} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$, the classification function is

$$f(\mathbf{x}) = \beta_0 + \boldsymbol{\beta}^{\mathrm{T}} \mathbf{x} = \beta_0 + \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i^{\mathrm{T}} \mathbf{x}$$

and the decision rule is

$$\hat{y} = \begin{cases} 1 & f(\mathbf{x}_*) > 0 \\ -1 & f(\mathbf{x}_*) < 0. \end{cases}$$

Thus, the points for which $\alpha_i > 0$ are the *support vectors* while the points $(\mathbf{x}_i, y_i)$ for which $\alpha_i = 0$ are irrelevant for classification. This will become important for the generalization to support vector machines.

Nonseparable Case

[Picture]

In the *nonseparable case*, a separating hyperplane does not exist. We seek a hyperplane that *almost* separates the data, using a *soft margin*.

7.2 Support Vector Classifier

Maximal margin classifiers can be very sensitive to the addition of a single new data point.

[Picture (Fig 9.5 ISLR)]

We want a method that is more robust against individual observations and can successfully classify *most* of the training data.

The *support vector classifier* (SVC) or *soft margin classifier* allows some of the data to fall on the wrong side of the classifier (soft implies that some data violate the classification rule). The optimization problem resulting in the SVC is

maximize$_{\beta_0,\beta_1,\ldots,\beta_p,\varepsilon_1,\ldots,\varepsilon_n} M$

subject to $\sum_{j=1}^{p} \beta_j^2 = 1$

and $y_i(\beta_0 + \boldsymbol{\beta}^{\mathrm{T}}\mathbf{x}_i) \geq M(1-\varepsilon_i)$ for all $i = 1, \ldots, n$

and $\sum_{i=1}^{n} \varepsilon_i \leq C$ where $\varepsilon_i \geq 0$ for all $i = 1, \ldots, n$.

Note that if $\varepsilon_1 = \cdots = \varepsilon_n = 0$ this reduces to the maximal margin classifier (which doesn't exist in the nonseparable case).

As before, M is the width of the margin. Now, $\varepsilon_1, \ldots, \varepsilon_n$ are *slack variables*. If $\varepsilon_i = 0$ then $(\mathbf{x}_i, y_i)$ is correctly classified and is on the correct side of the margin. If $0 < \varepsilon_i < 1$ then the ith data point is correctly classified, but is on the wrong side of the margin. If $\varepsilon_i > 1$ then the ith point is misclassified. The constant C is a *tuning parameter* that controls the total amount of slack allowed. If C is close to zero, then our SVC will be close to the maximal margin classifier, whereas if $C \gg 0$ we tolerate many violations of the boundary and will also lead to large margins. C is typically chosen by cross-validation, and controls the bias-variance tradeoff (small C implies large bias, large C implies large variance).

[Picture]

Defining the Support Vectors

The optimization program can be rewritten in Lagrangian form

$$\frac{1}{2}\|\boldsymbol{\beta}\|_2^2 + C\sum_{i=1}^{n}\varepsilon_i - \sum_{i=1}^{n}\alpha_i(y_i(\beta_0 + \boldsymbol{\beta}^{\mathrm{T}}\mathbf{x}_i) - (1-\varepsilon_i)) - \sum_{i=1}^{n}\mu_i\varepsilon_i$$

which we minimize wrt $\beta_0, \boldsymbol{\beta}$ and $\varepsilon_1, \ldots, \varepsilon_n$. Setting derivatives equal to zero we get

$$\boldsymbol{\beta} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$
$$\sum_{i=1}^{n} \alpha_i y_i = 0$$
$$\alpha_i = C - \mu_i, i = 1, \ldots, n.$$

Its now clear that the solution has the form

$$f(\mathbf{x}) = \beta_0 + \boldsymbol{\beta}^{\mathrm{T}}\mathbf{x} = \beta_0 + \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i^{\mathrm{T}}\mathbf{x},$$

and is a linear classifier. The $(\mathbf{x}_i, y_i)$ such that $\alpha_i \neq 0$ are the *support vectors*. That only a few observations define the decision rule implies that the SVM is robust against outlying observations (whereas, e.g., LDA relies on the mean of *all* observations within a class and thus is non-robust).

[R example (SupportVectorClassifier.R)]

7.3 Support Vector Machines

Both the maximal margin classifier and the support vector classifier result in linear decision boundaries. A *support vector machine* (SVM) relaxes this and allows for nonlinear decision boundaries.

[Picture (ala Figure 9.8 ISLR)]

Transformations Revisited

In linear regression we saw that considering transformations of the covariates could lead to superior model fits. Rather than using

$$x_1, \ldots, x_p$$

we could use

$$\mathbf{x} = (x_1, \ldots, x_p, x_1^2, \ldots, x_p^2)^{\mathrm{T}}.$$

Consider the effect of doing this in the following example:

Example 6. *Suppose we have data* $(\mathbf{x}_i, y_i)$ *for* $i = 1, 2, 3$ *with* $p = 2$ *predictors. That is,*

$$\mathbf{x}_1 = \begin{pmatrix} x_{11} \\ x_{12} \end{pmatrix}, \quad \mathbf{x}_2 = \begin{pmatrix} x_{21} \\ x_{22} \end{pmatrix}, \quad \mathbf{x}_3 = \begin{pmatrix} x_{31} \\ x_{32} \end{pmatrix}$$

and $y_i = \pm 1$. *[Picture] Then the classification function is, for* $\mathbf{x} = (x_1, x_2)^{\mathrm{T}}$,

$$\begin{aligned} f(\mathbf{x}) &= \beta_0 + \alpha_1 y_1 \mathbf{x}_1^{\mathrm{T}} \mathbf{x} + \alpha_2 y_2 \mathbf{x}_2^{\mathrm{T}} \mathbf{x} + \alpha_3 y_3 \mathbf{x}_3^{\mathrm{T}} \mathbf{x} \\ &= \beta_0 + \alpha_1 y_1 (x_{11} x_1 + x_{12} x_2) + \alpha_2 y_2 (x_{21} x_1 + x_{22} x_2) + \alpha_3 y_3 (x_{31} x_1 + x_{32} x_2) \\ &= \beta_0 + (\alpha_1 y_1 x_{11} + \alpha_2 y_2 x_{21} + \alpha_3 y_3 x_{31}) x_1 + (\alpha_1 y_1 x_{12} + \alpha_2 y_2 x_{22} + \alpha_3 y_3 x_{32}) x_2 \end{aligned}$$

which is of the format

$$f(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

so that $f(\mathbf{x}) = 0$ *defines a hyperplane. [rewrite* $x_1 = x$ *and* $x_2 = y$ *for* (x, y) *representation that "feels" better]*

Now, what happens if we amend the data vectors with transformations of the original predictors, say by including quadratic terms:

$$\mathbf{x}_1 = \begin{pmatrix} x_{11} \\ x_{12} \\ x_{11}^2 \\ x_{12}^2 \\ x_{11} x_{12} \end{pmatrix}, \quad \mathbf{x}_2 = \begin{pmatrix} x_{21} \\ x_{22} \\ x_{21}^2 \\ x_{22}^2 \\ x_{21} x_{22} \end{pmatrix}, \quad \mathbf{x}_3 = \begin{pmatrix} x_{31} \\ x_{32} \\ x_{31}^2 \\ x_{32}^2 \\ x_{31} x_{32} \end{pmatrix}.$$

Just by writing out, long hand, the inner products, we can get find that

$$\begin{aligned} f(\mathbf{x}) &= \beta_0 + \alpha_1 y_1 \mathbf{x}_1^{\mathrm{T}} \mathbf{x} + \alpha_2 y_2 \mathbf{x}_2^{\mathrm{T}} \mathbf{x} + \alpha_3 y_3 \mathbf{x}_3^{\mathrm{T}} \mathbf{x} \quad \textit{[as before]} \\ &= \beta_0 + (\textit{some coefficient}) x_1 + (\textit{some coefficient}) x_2 + (\textit{some coefficient}) x_1^2 + \\ &\qquad (\textit{some coefficient}) x_2^2 + (\textit{some coefficient}) x_1 x_2. \end{aligned}$$

This is an equation of the form $f(x, y) = a + bx + cy + dx^2 + ey^2 + fxy$, *and setting* $f(\mathbf{x}) = 0$ *then yields a* nonlinear *classification region that is quadratic.*

In general, with this motivation, we can amend each $\mathbf{x}_i$ with a bunch of transformations of the $x_{i1}, \ldots, x_{ip}$ predictors, yielding nonlinear classification regions. The solution is still calculated from the algorithm

$\min_{\beta_0,\beta_1,\ldots,\beta_{\#\,of\,predictors},\varepsilon_1,\ldots,\varepsilon_n} M$

subject to $y_i(\beta_0 + \boldsymbol{\beta}^{\mathrm{T}}\mathbf{x}) \geq M(1-\varepsilon_i)$

and $\sum_{i=1}^{n} \varepsilon_i \leq C$, $\|\boldsymbol{\beta}\|_2^2 = 1$, $\varepsilon_i \geq 0$.

The key to the support vector machine, next, is recognizing that (regardless of which transformations we insert into $\mathbf{x}$) the classification function can always be written

$$f(\mathbf{x}) = \beta_0 + \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i^{\mathrm{T}}\mathbf{x} \tag{7}$$

which depends on $\mathbf{x}_1, \ldots, \mathbf{x}_n$ *only through* the inner product

$$\langle \mathbf{x}_i, \mathbf{x} \rangle = \mathbf{x}_i^{\mathrm{T}}\mathbf{x}.$$

The Support Vector Machine

The generalization of the SVC requires the use of a *positive definite kernel.*

Definition 7. *We call a function* $k(\mathbf{x},\mathbf{y}) : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$ *a* positive definite function *if, for any* $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^p$ *and* $a_1, \ldots, a_n \in \mathbb{R}$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0.$$

The *support vector machine* replaces (7) with

$$f(\mathbf{x}) = \beta_0 + \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}).$$

As previously, the *support vectors* are those for which $\alpha_i \neq 0$.

There are many options for kernels, some of the most popular are:

- Polynomial: $k(\mathbf{x}_i, \mathbf{x}) = (1 + \mathbf{x}_i^{\mathrm{T}}\mathbf{x})^d$
- Radial: $k(\mathbf{x}_i, \mathbf{x}) = e^{-a\|\mathbf{x}_i - \mathbf{x}\|^2}$.

Example 8. *Note what happens with a polynomial kernel of degree $d = 1$. If $d = 2$ then it's not hard to see that, in the case of two predictors with $\mathbf{x} = (x_1, x_2)^{\mathrm{T}}$,*

$$\begin{aligned}(1 + \mathbf{x}_i^{\mathrm{T}}\mathbf{x})^2 &= (1 + x_{i1}x_1 + x_{i2}x_2)^2 \\ &= 1 + 2x_{i1}x_1 + 2x_{i2}x_2 + x_{i1}^2x_1^2 + x_{i2}^2x_2^2 + 2x_{i1}x_{i2}x_1x_2\end{aligned}$$

which is quadratic in $\mathbf{x}$.

If we had used the trick from before in Example 6, building support vector classifier (i.e., linear decision boundary) with transformed features features $\mathbf{x} = (\sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)^{\mathrm{T}}$ *then our decision boundary would use weighted combinations of*

$$\mathbf{x}_i^{\mathrm{T}}\mathbf{x} = 2x_{i1}x_1 + 2x_{i2}x_2 + x_{i1}^2x_1^2 + x_{i2}^2x_2^2 + 2x_{i1}x_{i2}x_1x_2.$$

Thus, a polynomial kernel yields the same nonlinear decision boundary shape as if we engineered *some extra polynomial features and did a standard "linear" classifier. It is key to note that the linear classifier does not know that, for example, the third "feature" x_1^2, is at all related to the first feature $\sqrt{2}x_1$, it treats them as separate pieces of information, so* thinks *that we are working in a 5-dimensional feature space, rather than 2-dimensional.*

Example 9. *Consider the same setup as the first part of Example 6 but now with a SVM and k being a radial kernel. We have*

$$\begin{aligned}f(\mathbf{x}) &= \beta_0 + \alpha_1 k(\mathbf{x}_1, \mathbf{x}) + \alpha_2 k(\mathbf{x}_2, \mathbf{x}) + \alpha_3 k(\mathbf{x}_3, \mathbf{x}) \\ &= \beta_0 + \alpha_1 e^{-a\|\mathbf{x}_1 - \mathbf{x}\|^2} + \alpha_2 e^{-a\|\mathbf{x}_2 - \mathbf{x}\|^2} + \alpha_3 e^{-a\|\mathbf{x}_3 - \mathbf{x}\|^2}.\end{aligned}$$

[What does $f(x) = \exp(-|x_1 - x|^2)$ look like in one dimension? In two dimensions $\exp(-\|\mathbf{x}_1 - \mathbf{x}\|^2)$ is the same thing, but is a circular bump function centered at $\mathbf{x}_1$] Thus, $f(\mathbf{x})$ is simply the sum of a weighted set of squared exponential bump functions centered at the data features $\mathbf{x}_1, \mathbf{x}_2$ and $\mathbf{x}_3$ [picture with $y_1 = y_2 = +1$ and $y_3 = -1$].

In any case, the decision rule is still

$$\hat{y} = \operatorname{sgn}\left(\beta_0 + \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x})\right).$$

You might ask "why can we just swap in a kernel $k(\mathbf{x}_i, \mathbf{x})$ instead of a product like $\mathbf{x}_i^{\mathrm{T}}\mathbf{x}$ with some clever transformations of the features appended inside $\mathbf{x}$? You can't just choose to put a kernel in anywhere you want!" That is a fair point. It turns out that Mercer's Theorem is the key link. We will get to a technical definition of this in the Gaussian process section, but for now the important thing is to note that *any* positive definite function (AKA kernel) can be written

$$k(\mathbf{v}, \mathbf{w}) = \sum_{i=1}^{\infty} \lambda_i \phi_i(\mathbf{v}) \phi_i(\mathbf{w}) \tag{8}$$

for some nonnegative coefficients $\lambda_1 \geq \lambda_2 \geq \cdots \geq 0$ and functions $\{\phi_i(\mathbf{x})\}$. So, if we were really clever and knew what these mysterious $\{\phi_i(\mathbf{x})\}$ functions were we could just parameterize a "new" feature vector full of these transformations, e.g.,

$$\mathbf{x}_{i,new} = (\sqrt{\lambda_1}\phi_1(\mathbf{x}_i), \sqrt{\lambda_2}\phi_2(\mathbf{x}_i), \ldots, \sqrt{\lambda_N}\phi_N(\mathbf{x}_i))^{\mathrm{T}}$$

for some really big N. Then, if we just do a regular "linear" support vector classifier we end up with

$$\begin{aligned} f(\mathbf{x}) &= \beta_0 + \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_{i,new}^{\mathrm{T}} \mathbf{x}_{new} \\ &= \beta_0 + \sum_{i=1}^{n} \alpha_i y_i \left(\sum_{j=1}^{N} \lambda_j \phi_j(\mathbf{x}_i) \phi_j(\mathbf{x}) \right) \qquad \text{[note these are the original } p\text{-variate } \mathbf{x}_i\text{s]} \\ &\approx \beta_0 + \sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) \end{aligned}$$

where the last approximation comes from approximating the infinite sum (10) with a finite sum.

[R example (SupportVectorMachine.R)]

Which Classifier Should I Choose?

Depends on your goal:

- Inference: logistic or probit regression, discriminant analysis
- Prediction: discriminant analysis, KNN, SVM

SVMs With More Than Two Classes

It is difficult to extend the methodology of SVMs to more than two classes. Suppose there are $K > 2$ classes; the two most popular approaches are:

- *One vs. One Classification*: build $\binom{K}{2}$ SVMs for comparing all pairwise classes, and, to predict for a new set of covariates, tally the class outcomes for each of the $\binom{K}{2}$ SVMs and favor the one with the highest tally count.
- *One vs. All Classification*: fit K different SVMs for comparing the kth class against everything else (where everything else is coded the same, e.g., -1), $k = 1, \ldots, K$. Call the estimated parameters $\beta_{0k}, \beta_{1k}, \ldots, \beta_{pk}$ for the kth class. Then to predict for a new covariate $\mathbf{x}^*$, use the class for which $\beta_{0k} + \beta_{1k}x_1^* + \cdots + \beta_{pk}x_p^*$ is the greatest.

Support Vector Machines via Penalization

The support vector machine is the solution to the following constrained optimization problem:

$\min_{\beta_0,\boldsymbol{\beta}} \sum_{i=1}^n \varepsilon_i + \lambda\|\boldsymbol{\beta}\|_2^2$

subject to $y_i f(\mathbf{x}_i) \geq 1 - \varepsilon_i$ and $\varepsilon_i \geq 0$ for $i = 1, \ldots, n$

where $f(\mathbf{x}) = \beta_0 + \sum_{i=1}^n \beta_i k(\mathbf{x}_i, \mathbf{x})$. The constraint $y_i f(\mathbf{x}_i) \geq 1 - \varepsilon_i$ is the same as $\varepsilon_i \geq 1 - y_i f(\mathbf{x}_i)$ and minimizing $\sum_{i=1}^n \varepsilon_i + \lambda\|\boldsymbol{\beta}\|_2^2$, is equivalent to minimizing $\sum_{i=1}^n (1 - y_i f(\mathbf{x}_i)) + \lambda\|\boldsymbol{\beta}\|_2^2$, except that $\varepsilon_i \geq 0$, so it turns out the support vector machine is also the minimizer of

$$\sum_{i=1}^n (1 - y_i f(\mathbf{x}_i))_+ + \lambda\|\boldsymbol{\beta}\|_2^2$$

where minimization is over β_0 and $\boldsymbol{\beta}$, and where

$$(x)_+ = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$$

is known as the *hinge loss* function.

Since this problem results in the SVM solution, we see that the SVM falls into the form

$$\min_{\beta_0,\boldsymbol{\beta}} \sum_{i=1}^{n} L(y_i, \mathbf{x}_i, \beta_0, \boldsymbol{\beta}) + \lambda P(\boldsymbol{\beta})$$

where L is a *loss function* that measures "closeness to the data" and P is a *penalty* that measures "model complexity" (more next!). [Note the hinge loss penalty makes sense, check for $f \gg 0$ and what happens with $y = 1$, i.e., correct classification vs. $y = -1$, incorrect classification]

Let's return to the beginning of the classification work and reconsider logistic regression. It turns out the classifier from logistic regression also has the "loss plus penalty" format, but without the penalty. Comparing the loss functions between SVMs and logistic regression can yield some insights.

To derive, recall how logistic regression works: with $Y \in \{0, 1\}$ we model

$$P(Y = 1 \mid \mathbf{x}) = \frac{e^{f(\mathbf{x})}}{1 + e^{f(\mathbf{x})}}.$$

The resulting pmf for Y is

$$\begin{aligned} p(y) &= \left(\frac{e^{f(\mathbf{x})}}{1 + e^{f(\mathbf{x})}}\right)^{y} \left(1 - \frac{e^{f(\mathbf{x})}}{1 + e^{f(\mathbf{x})}}\right)^{1-y} \\ &= \left(\frac{1}{1 + e^{-f(\mathbf{x})}}\right)^{y} \left(\frac{1}{1 + e^{f(\mathbf{x})}}\right)^{1-y}. \end{aligned}$$

The only tricky thing here is that to connect to SVMs we need to change the pmf from $Y \in \{0, 1\}$ to $Y \in \{-1, +1\}$:

$$p(y) = \left(\frac{1}{1 + e^{-f(\mathbf{x})}}\right)^{\mathbb{1}_{[y=1]}} \left(\frac{1}{1 + e^{f(\mathbf{x})}}\right)^{\mathbb{1}_{[y=-1]}}.$$

Now take the negative log likelihood:

$$\begin{aligned} -\log p(y) &= \mathbb{1}_{[y=1]} \log(1 + e^{-f(\mathbf{x})}) + \mathbb{1}_{[y=-1]} \log(1 + e^{f(\mathbf{x})}) \\ &= \log(1 + e^{-yf(\mathbf{x})}) \end{aligned}$$

where the second equality just needs the sanity check.

Now the β_0 and $\boldsymbol{\beta}$ that define the logistic regression classification function $f(\mathbf{x})$ are MLEs, i.e., those that maximize the likelihood $p(y_1, \ldots, y_n)$, or equivalently those that maximize the log likelihood $\log p(y_1, \ldots, y_n)$, or equivalently those that minimize the negative log likelihood

$$\begin{aligned} -\log p(y_1, \ldots, y_n) &= -\log \prod_{i=1}^{n} p(y_i) \quad \text{(by independence)} \\ &= \sum_{i=1}^{n} \log(1 + e^{-y_i f(\mathbf{x}_i)}). \end{aligned}$$

To summarize, the SVM comes from the loss function

$$L(y, \mathbf{x}, \beta_0, \boldsymbol{\beta}) = (1 - yf(\mathbf{x}))_+$$

while logistic regression comes from the loss function

$$L(y, \mathbf{x}, \beta_0, \boldsymbol{\beta}) = \log(1 + \exp(-yf(\mathbf{x}))).$$

While the SVM coefficients will often be zeroed, logistic regression coefficients will almost always be nonzero.

[Picture]

Now, with all of this said, is there a probabilistic interpretation to SVMs? If the hinge loss $(1 - y_i f(\mathbf{x}_i))_+$ is thought of a negative log likelihood then the SVM model is equivalent to the following probability mass function:

$$p(y) \propto \exp(-(1 - yf(\mathbf{x}))_+) \quad y \in \{-1, +1\}$$

noting we need the $\propto$ since the two terms don't sum to one without standardization.

8 Model Inference

In this section we are interested in estimating uncertainty about statistics that do not allow for easy analytic formulations. For example, what is the uncertainty in the sample median? The sample mean has a nice formula,

$$\mathrm{Var}(\bar{X}) = \frac{\sigma^2}{n}$$

from which standard errors follow by $SE(\bar{X}) = \hat{\sigma}/\sqrt{n}$, and ensuing confidence intervals. But what is Var(median($X_1, \ldots, X_n$))? The bootstrap attempts to estimate the sampling distribution of a statistic, from which confidence intervals, standard errors or other inferences can be made.

Suppose we have an iid random sample of size n from some distribution F, call the samples $\mathbf{x} = (x_1, \ldots, x_n)^{\mathrm{T}}$. We are interested in estimating a parameter from the distribution F,

$$\theta = T(F).$$

For example, X is some random variable with finite mean, we might be interested in that mean:

$$\mu = \int x \mathrm{d}F(x).$$

Given data, we have the empirical cumulative distribution function

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{[x_i \le x]}(x)$$

[picture of empirical cdf] from which we can use, e.g.,

$$T(F_n) = \int x \, \mathrm{d}F_n(x) = \frac{1}{n}\sum_{i=1}^{n} x_i = \bar{x}.$$

This type of estimator is called a *plug-in estimator* because we are plugging in the empirical cdf (known) for its true (unknown) version.

8.1 Jackknife

Suppose we have an iid random sample of size n from some distribution F, call the samples $\mathbf{x} = (x_1, \ldots, x_n)^{\mathrm{T}}$. To estimate θ we have some *statistic* based on the data,

$$\hat{\theta} = s(\mathbf{x}) \quad (= T(F_n)).$$

We would like to understand the uncertainty and bias in our estimator $\hat{\theta}$, but this may be difficult analytically. The *jackknife* is one approach to estimating the bias and standard error of $\hat{\theta}$, and works best for smooth functions of F such as the mean or other moments.

Recall that statistical bias of an estimator $\hat{\theta}$ that is trying to estimate the population parameter θ is defined as

$$\text{bias} = \mathbb{E}_F(\hat{\theta}) - \theta.$$

Example 10. *If $X_1, \dots, X_n$ are iid from some distribution with mean μ and variance σ^2 (note these don't have to be normal!) then we have*

$$\text{bias}(\bar{X}) = \mathbb{E}(\bar{X}) - \mu = \mu - \mu = 0,$$

so the sample mean is unbiased. However, one estimator of the sample variance (which happens to be the maximum likelihood estimate in the case that the X_is are normal) has some bias:

$$\begin{aligned}\text{bias}(\hat{\sigma}^2) &= \mathbb{E}\left(\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2\right) - \sigma^2 \\ &= \frac{n-1}{n}\sigma^2 - \sigma^2 \\ &= -\frac{1}{n}\sigma^2,\end{aligned}$$

so that this estimator is a little biased.

Define the *leave-one-out observation*

$$\mathbf{x}_{(i)} = (x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n)^{\mathrm{T}}$$

which is a vector of length $n-1$. Define

$$\hat{\theta}_{(i)} = s(\mathbf{x}_{(i)})$$

to be the *ith jackknife replication of $\hat{\theta}$.* The jackknife estimate of bias is simply

$$\widehat{\text{bias}} = (n-1)(\hat{\theta}_{(\cdot)} - \hat{\theta})$$

where

$$\hat{\theta}_{(\cdot)} = \frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_{(i)}.$$

We also have the jackknife estimate of standard error is

$$\widehat{\mathrm{SE}} = \left(\frac{n-1}{n}\sum_{i=1}^{n}(\hat{\theta}_{(i)} - \hat{\theta}_{(\cdot)})^2\right)^{1/2}.$$

Note the jackknife estimate of standard error looks a lot like a regular standard error, except the factor in front is $(n-1)/n$ instead of $1/(n-1)$ or $1/n$ – this is because the jackknife deviations

$$\hat{\theta}_{(i)} - \hat{\theta}_{(\cdot)}$$

tend to be very close together because only a single data point is being deleted for each deviation, similarly for the bias estimate. We can then set up approximate 95% confidence intervals as

$$\hat{\theta} \pm 2\widehat{\mathrm{SE}}.$$

As an aside, now that we have $\widehat{\text{bias}}$, we could potentially bias-correct our estimator $\hat{\theta}$ such as

$$\hat{\theta} - \widehat{\text{bias}}.$$

However, this estimator of bias is also random, and tends to be quite variable, so using it introduces extraneous variability to our estimate of θ which may in turn ruin the estimator's mean squared error. A little bias is okay, as we've seen, especially if we can drastically reduce an estimator's variability.

The jackknife fails quite badly for nonsmooth statistics such as quantiles like the median. The bootstrap is a more general approach to estimating things like standard errors.

8.2 Nonparametric Bootstrap

We saw that the jackknife estimated statistical bias and standard errors by subsampling the observation data vector, deleting one observation at a time. The *bootstrap* is a more general approach to estimating statistic characteristics by sampling, and works for nonlinear statistics such as the maximum or median.

We resample the data vector $\mathbf{x}$ with replacement: call

$$\mathbf{x}^* = (x_1^*, x_2^*, \ldots, x_n^*)^{\mathrm{T}}$$

a bootstrap dataset which consists of members of the original vector $\mathbf{x}$, but that are sampled with replacement. So, x_1 may appear no times, x_2 three times, x_3 once, etc.

Example 11. *Suppose* $\mathbf{x} = (0, 12, 1/2)^{\mathrm{T}}$. *Then one possible bootstrap dataset would be* $\mathbf{x}^* = (0, 0, 12)^{\mathrm{T}}$, *another is* $\mathbf{x}^* = (1/2, 1/2, 1/2)^{\mathrm{T}}$. *In this case there are* $3^3 = 27$ *possible bootstrap datasets.*

For an original observation of length n, there are n^n possible bootstrap samples. One way to think about this is that $\mathbf{x}$ is a sample from F, while $\mathbf{x}^*$ is a sample from F_n.

Corresponding to each bootstrap dataset $\mathbf{x}^{*b}, b = 1, \ldots, n^n$, we have the bootstrap replication of $\hat{\theta}$ is

$$\hat{\theta}^*(b) = s(\mathbf{x}^{*b}),$$

so, for instance, if s is the sample mean then

$$s(\mathbf{x}^{*b}) = \frac{1}{n}\sum_{i=1}^{n} x_i^{*b}.$$

The idea is that the *bootstrap replicates* $\{\hat{\theta}^*(b)\}_{b=1}^{n^n}$ can be thought of as samples from the sampling distribution of $\hat{\theta}$, and then used to calculate statistics about $\hat{\theta}$ like the standard error. Of course, n^n is way too big to actually re-calculate $\hat{\theta}$ that many times, so in practice we *randomly generate* bootstrap samples, and thus bootstrap replicates, from which we then estimate statistics of interest.

[Picture]

For example, we can use the bootstrap to estimate the standard error of $\hat{\theta}$ using the following algorithm:

1. Select B independent bootstrap samples $\mathbf{x}^{*1}, \ldots, \mathbf{x}^{*B}$ each of length n, representing sampling with replacement from $\mathbf{x}$.

2. Evaluate the bootstrap replication for each sample
$$\hat{\theta}^*(b) = s(\mathbf{x}^{*b}) \qquad b = 1, \ldots, B$$
3. Estimate the standard error by the sample standard deviation of the B replicates,
$$\widehat{SE}_B = \left(\frac{1}{B-1} \sum_{b=1}^{B} (\hat{\theta}^*(b) - \hat{\theta}^*(\cdot))^2 \right)^{1/2}$$
where $\hat{\theta}^*(\cdot) = B^{-1} \sum_b \hat{\theta}^*(b)$ is the sample mean of the bootstrap replicates.

How big should B be? Well, in principle the best we can do is $B = n^n$, but of course for any reasonable modern dataset this is not feasible. The basic idea is more is better, but at least a couple hundred is a good starting place.

8.3 Parametric Bootstrap

The only difference between the nonparametric bootstrap from above, is that instead of sampling $\mathbf{x}^*$ from F_n (i.e., resampling from the observed data), we sample $\mathbf{x}^*$ from a parametric distribution $\hat{F}$, which of course is often estimated from the data. For example, suppose we have some data $\mathbf{x}$ that we *think* might come from an exponential distribution, $\text{Exp}(\lambda)$ with pdf
$$f(x) = \lambda \exp(-x\lambda) \qquad x > 0$$
and mean $1/\lambda$. An estimator for λ might be $1/\bar{x} = \hat{\lambda}$. To assess the uncertainty in $\hat{\lambda}$ we could simulate n iid samples from $\text{Exp}(\hat{\lambda})$, and do this B times to get $\mathbf{x}^{*1}, \ldots, \mathbf{x}^{*B}$. Taking the sample standard deviation of $\{\mathbf{x}^{*b}\}$ will be an estimate of $SE(\hat{\lambda})$.

[R example (Bootstrap.R)]

8.4 Bootstrap for Regression

Suppose we are back in a regression situation with $\mathbf{z}_1 = (\mathbf{x}_1, y_1), \ldots, \mathbf{z}_n = (\mathbf{x}_n, y_n)$ being observations from the model
$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}.$$
There are at least two ways to approach the bootstrap approach to generating standard errors for the OLS estimator $\hat{\boldsymbol{\beta}}$: bootstrapping pairs or residuals.

Bootstrapping Pairs

To bootstrap pairs, we complete the following algorithm for $b = 1, \ldots, B$:

1. Select independent bootstrap samples $\mathbf{z}^{*1}, \ldots, \mathbf{z}^{*n}$ that are each vectors from the set $\{\mathbf{z}_1, \ldots, \mathbf{z}_n\}$, with replacement.
2. Compute the OLS estimator of $\boldsymbol{\beta}$ based on pairs $\{\mathbf{z}^{*i}\}_{i=1}^n$, call it $\hat{\boldsymbol{\beta}}^*(b)$.

We can then estimate the covariance matrix of $\boldsymbol{\beta}$

$$\widehat{\mathrm{Var}}_B(\hat{\boldsymbol{\beta}}) = \frac{1}{B-1}\sum_{b=1}^{B}(\hat{\boldsymbol{\beta}}^*(b) - \hat{\boldsymbol{\beta}}^*(\cdot))(\hat{\boldsymbol{\beta}}^*(b) - \hat{\boldsymbol{\beta}}^*(\cdot))^{\mathrm{T}}$$

where $\hat{\boldsymbol{\beta}}^*(\cdot) = B^{-1}\sum_b \hat{\boldsymbol{\beta}}^*(b)$ is the sample mean of the bootstrap replicates. The standard errors for each individual parameter can be approximated as the square roots of the diagonal of $\widehat{\mathrm{Var}}_B(\hat{\boldsymbol{\beta}})$.

Bootstrapping Residuals

Another approach is to bootstrap the residuals. Start with the original model

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

and compute the OLS estimator $\hat{\boldsymbol{\beta}}$ and associated estimated residuals,

$$\hat{\boldsymbol{\varepsilon}} = \mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}},$$

call the ith estimated residual $\hat{\varepsilon}_i$.

Bootstrapping residuals essentially boils down to generating new response values from the original residuals. To bootstrap residuals, we complete the following algorithm for $b = 1, \ldots, B$:

1. Generate independent bootstrap samples of the residuals, $\varepsilon_1^*, \ldots, \varepsilon_n^*$ from $\hat{\varepsilon}_1, \ldots, \hat{\varepsilon}_n$, with replacement.

2. Generate
$$y_i^* = \mathbf{x}_i^{\mathrm{T}}\hat{\boldsymbol{\beta}} + \varepsilon_i^*$$
for each $i = 1, \ldots, n$.

3. Compute the OLS estimator of $\boldsymbol{\beta}$ based on pairs $\{(\mathbf{x}_i, y_i^*)\}_{i=1}^n$, call it $\hat{\boldsymbol{\beta}}^*(b)$.

We can then estimate the covariance matrix of $\boldsymbol{\beta}$ in the same way as before,
$$\widehat{\mathrm{Var}}_B(\hat{\boldsymbol{\beta}}) = \frac{1}{B-1}\sum_{b=1}^{B}(\hat{\boldsymbol{\beta}}^*(b) - \hat{\boldsymbol{\beta}}^*(\cdot))(\hat{\boldsymbol{\beta}}^*(b) - \hat{\boldsymbol{\beta}}^*(\cdot))^{\mathrm{T}}$$
where $\hat{\boldsymbol{\beta}}^*(\cdot) = B^{-1}\sum_b \hat{\boldsymbol{\beta}}^*(b)$ is the sample mean of the bootstrap replicates. The standard errors for each individual parameter can be approximated as the square roots of the diagonal of $\widehat{\mathrm{Var}}_B(\hat{\boldsymbol{\beta}})$.

Which to Bootstrap: Pairs or Residuals?

Which should we resample, pairs or residuals? It depends on how much we believe the linear model assumption that the distribution of ε_i does not depend on i – if the errors are heteroskedastic, for instance, then we should not resample residuals since that breaks the relationship between $\mathbf{x}_i$ and ε_i. Generally, bootstrapping pairs is less sensitive to assumptions than bootstrapping residuals.

Parametrically

Yet another way to do bootstrap for regression is to assume that $\boldsymbol{\varepsilon}$ come from a particular distribution, say $N(0, \sigma^2)$ (of course we can diagnostically check this using a Q-Q plot on $\hat{\boldsymbol{\varepsilon}}$, say). Then we follow the outline of bootstrapping the residuals where in step 1, we generate ε_i^* from $N(0, \hat{\sigma}^2)$ instead of resampling from the estimated residuals.

[R example (BootstrapRegression.R)]

9 Tree-Based Methods

Tree-based methods work for both regression and classification. They act by *stratifying* or *segmenting* the predictor space into regions and use a separate simple predictor in each region. These are known as *decision tree* methods. Trees are useful for interpretation, but are not necessarily as good at prediction as the methods we have already discussed. However, we will see that some clever insights from prior chapters will lead to generalized tree-based approaches proving powerful predictive models that compete with the best modern techniques.

A few factoids:

- Trees are easy to explain (and interpret)
- Trees have easy graphical depictions
- Trees can handle qualitative predictors without the need for dummy variables
- Trees will typically perform very poorly in terms of prediction compared to a linear model

This section discusses the basics of the CART (classification and regression trees) implementation of trees.

9.1 Regression Trees

A regression tree just splits the predictor space into regions, and uses the average response within each region as the predictor.

Example 12. *If Y is rating of a movie (out of 10) and X is the budget (in dollars), then a simple regression tree might predict*

$$\hat{y} = \begin{cases} 7.03, & x < 10250 \\ 6.07, & x > 10250. \end{cases}$$

The line $x = 10250$ defines the two regions in this case.

Example 13. *If Y is InstructorOverall in the CU FCQ database and we use two predictors X_1 =Challenge and X_2=PriorInterest, then we fit a regression tree and found*

- *If Challenge* < 4.76
  - *and Challenge* < 3.98 *then* $\hat{Y} = 4.56$
  - *and Challenge* ≥ 3.98 *then* $\hat{Y} = 5.03$
- *Else if Challenge* ≥ 4.76
  - *and PriorInterest* < 4.99 *then* $\hat{Y} = 5.25$
  - *and PriorInterest* ≥ 4.99 *then* $\hat{Y} = 5.58$*.*

[Picture in 2d and associated tree ala Figure 8.3 of ISLR]

For continuous response Y and p predictors $\mathbf{X} = (X_1, \ldots, X_p)^{\mathrm{T}}$, the basic steps to build a regression tree are:

(1) Divide predictor space into M disjoint regions $R_1, \ldots, R_M$.

(2) To predict a new response in R_m, use the the mean of all observations in R_m.

The statistical model can still be written

$$Y = f(\mathbf{X}) + \varepsilon,$$

where we set

$$f(\mathbf{X}) = \sum_{m=1}^{M} c_m \mathbb{1}_{[\mathbf{X} \in R_m]}$$

to be the constant c_m in each region R_m.

Given some data $(y_1, \mathbf{x}_1), \ldots, (y_n, \mathbf{x}_n)$, if we choose to use

$$\sum_{i=1}^{n} (y_i - f(\mathbf{x}_i))^2$$

i.e., the regular sum of squares, as our objective criterion, then it's not too hard to show that

$$\hat{c}_m = \text{ave}\{y_i \,|\, \mathbf{x}_i \in R_m\}$$

for all m (that is, just take the average value of the y_i such that $\mathbf{x}_i$ is in the mth region).

So, how do we choose regions $\{R_m\}$? Note that we could use weird shapes for $\{R_m\}$ to get a lot of flexibility, but using hyperrectangles/boxes is easier for interpretation, implementation and computation, and is the standard approach. Finding these splits relies on *recursive binary splitting*, a *top-down greedy* approach.

[Top-down = begins at the top of the tree, lopping all observations together]

[Greedy = at each step of building, use the best split without looking into the future]

The first step is, for a given m and splitting cutoff s, form half-planes $R_1(m,s) = \{X | X_m < s\}$ and $R_2(m,s) = \{X | X_m \geq s\}$, and calculate

$$\min_{c_1} \sum_{\mathbf{x}_i \in R_1(m,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{\mathbf{x}_i \in R_2(m,s)} (y_i - c_2)^2$$

which is achieved at

$$\hat{c}_k = \text{ave}\{y_i \,|\, \mathbf{x}_i \in R_k(m,s)\} \quad \text{for} \quad k = 1, 2.$$

Then the final choice for first variable/split combination is the pair (m, s) that solves

$$\min_{m,s} \left[\min_{c_1} \sum_{\mathbf{x}_i \in R_1(m,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{\mathbf{x}_i \in R_2(m,s)} (y_i - c_2)^2 \right].$$

Because this involves only simple averages for each choice of m and s, the minimizer can be found very quickly.

The next step is to repeat this process separately within *each* of the new regions $R_1(m,s)$ and $R_2(m,s)$. Such a procedure is iterated until some stopping criterion is reached, e.g., no region contains more than 5 observations.

[Thought experiment: what if we do not give a stopping criterion? Can graph a simple example with 2-3 observation pairs (y, x) and get a perfect fit eventually]

This procedure will overfit the data, the bias may be low, but at the cost of high variance. Thus, we could a very large tree T_0 and *prune* back to obtain a *subtree*. We could do this by *cost-complexity pruning*, also known as *weakest link pruning*. Rather than considering every possible subtree, consider a sequence of trees indexed by a tuning parameter $\alpha \geq 0$. In particular, for each α there is a subtree $T \subset T_0$ that minimizes

$$\sum_{m=1}^{|T|} \left(\sum_{i|\mathbf{x}_i \in R_m} (y_i - \hat{c}_m)^2 \right) + \alpha |T|.$$

Here, $|T|$ is the number of terminal nodes in the tree, and R_m is the rectangle corresponding to the mth terminal node with corresponding fitted value $\hat{c}_m$.

Note this is just another loss + penalty idea, where the penalty is now on the number of nodes of the tree. If $\alpha = 0$ then we will favor the full tree, while as $\alpha \to \infty$ we shrink toward smaller trees. Interestingly one can show that for each α, there is a unique smallest subtree T_α that minimizes this criterion. The smoothing parameter α can be chosen by cross-validation.

[R example (RegressionTrees.R)]

9.2 Bagging

Trees suffer from *high variance*, that is, if the data are randomly split into two pieces and models are fit on both, the two trees will typically be quite different.

Bootstrap aggregation or *bagging* reduces the variability of a tree by bootstrap resampling the training data into B training sets, calculating a separate tree for each, and averaging the predictors. More formally, for $b = 1, \ldots, B$,

- Generate a bootstrap sample, with replacement, of the training data, $\{(\mathbf{x}_i^*, y_i^*)\}_{i=1}^n$.
- Fit a tree on the bootstrap resampled data, $\hat{f}^b(\mathbf{x})$.

The final prediction is the sample average of the individual trees:

$$\hat{f}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^b(\mathbf{x}).$$

This approach is referred to as an *ensemble* approach, because an ensemble of individual models are averaged to produce a single estimate. Ensemble models tend to perform very well in practice and are some of the most competitive supervised learning approaches out there.

Bagged trees tend to perform much better in out-of-sample prediction, but because we are using many trees, it is not so obvious how to determine which features are playing prominent roles.

Various so-called *variable importance* measures have been proposed for ensemble-based methods. For regression trees, recording the total amount that the RSS is reduced due to splits over a given predictor, which are then averaged over all trees can give a sense for the importance of that predictor. Large reductions indicate important predictors. This is sometimes referred to as measuring *node impurity.*

Another approach uses permutation of features. This requires the notion of the *out of bag (OOB)* sample. Recall that when the bth tree is grown, each tree is fit on a bootstrapped dataset, so (potentially) some data points were not used to fit that particular tree – these are the OOB samples for that tree. We do two things: first, produce the fitted values of the OOB samples under the fitted tree (this has a cross-validation flavor) and record the mean squared error. The next step seems bizarre at first blush: take the jth feature for the OOB samples, and permute them among all OOB samples (that is, randomly shuffle them between the observations). Then, for the (jth) permuted feature, re-predict the OOB samples and record the mean squared error. Do this for all $j = 1, \ldots, p$ features. Record the decrease in accuracy for each feature for the bth tree, and then do this for all $b = 1, \ldots, B$ trees (each of which will (probably) have different OOB samples). Finally, average the decrease in accuracy for each permuted feature over all trees.

The idea with permutation is that if variable j has nothing to do with y but variable k is really important, then we should not see much reduction in MSE by permuting j but will see a big hit to MSE by permuting k. You can imagine this in a silly example: if y is snow cover percentage and x_k is elevation, we know that elevation has a big impact on snow (higher elevations usually have more snow) so messing around with locations' elevations would probably screw up our predictions, while if x_j is how many tourists visited Las Vegas that year, that probably has negligible effect on how much snow there is in the mountains, so switching this year's tourist rate with another year's won't help us understand how much snow there is (okay, maybe there is something to do with airline exhaust and cloud seeding, etc., but you get the point).

[R example (Bagging.R)]

9.3 Classification Trees

A *classification tree* is the same thing as a regression tree, but where the response is categorial. Rather than using the simple average of the response in each region R_m, our prediction is the *most commonly occurring class.* Suppose we have K classes. In a node m representing a region R_m with N_m observations let

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{\mathbf{x}_i \in R_m} \mathbb{1}_{[y_i=k]}$$

denote the empirical probability of class k in region R_m. We classify the observations into node m to class $k(m) = \text{argmax}_k \hat{p}_{mk}$, i.e., the majority class in node m.

Instead of residual sum of squares, we need a goodness-of-fit criterion $Q_m(T)$ to plug into the cost-complexity objective

$$\sum_{m=1}^{|T|} Q_m(T) + \alpha|T|.$$

Two common measures are the *Gini index*,

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) = 1 - \sum_{k=1}^{K} \hat{p}_{mk}^2$$

Note that G is small if all $\hat{p}_{mk}$s are close to zero or one, and thus the Gini index is a measure of node *purity.* The *cross-entropy* or *deviance* index is

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$

and is also close to zero if $\hat{p}_{mk}$ are always close to zero or one.

We can also bag classification trees, but here, rather than averaging we use the *majority vote*, that is, the classified value that is most common amongst all B trees. As previously, bagged classification trees are difficult to interpret. However, *variable importance* can still be interpreted as the reduction in RSS (or, say Gini index) due to splits over a given predictor averaged over B trees.

[R example (ClassificationTrees.R)]

9.4 Caveats

If a given predictor is categorical with q classes, there are $2^{q-1} - 1$ possible splits of the q values into two groups, which becomes computationally prohibitive for large q. Moreover, partitioning algorithms tend to favor categorical variables with large q as the possible number of splits grows exponentially, so such predictors should be avoided. Importance measures such as node purity tends to be biased to favor such categorical features, whereas permutation-based importance measures are not.

Trees are also notoriously high variance: if a single observation is removed the optimal tree can drastically change; ensemble-based methods avoid this pitfall. Finally, trees are piecewise constant predictors, and are not smooth; multivariate adaptive regression splines (MARS) attempt to overcome this problem.

9.5 Random Forests

Random forests follow the exact same approach as bagging trees, by building an ensemble of individual trees that are each trained on bootstrapped datasets. The main difference is that for each tree, at each individual split, a subset of *features* are randomly selected to consider, rather than trying all possible features. If features are highly correlated, this has the effect of decorrelating them, leading to more independent trees.

If there are p total possible predictors, for each tree at each split, a random sample of $m < p$ are considered, where typically $m \approx \sqrt{p}$ for classification forests, and $m \approx p/3$ for regression forests. Variable importance measures follow from the bagging trees approach.

Random forests tend to do poorly if there are many predictors, but only a small subset are actually related to the response (you can imagine that in those cases the "important" features may often fail to be selected in the split subsamples leading to splits over irrelevant features).

[R example (RandomForests.R)]

9.6 Boosting

In bagging we fit many trees in parallel by resampling original dataset using the bootstrap. *Boosting* grows an ensemble of regression trees, but instead does so *sequentially*, where each new tree is grown using information from previously grown trees. Boosting does not use the bootstrap, but rather modifies the original dataset at each stage. The essential idea is to fit a simple model first, and then look at the errors of that model – fit a new model to the residuals of the first model, in essence *fixing* places where the original model fit was poor.

9.6.1 AdaBoost.M1

AdaBoost is a popular boosting method for classification problems (that can be extended to regression problems). We introduce the basic AdaBoost.M1 algorithm which is a classification approach for categorical data $Y \in \{-1, +1\}$ (it can be extended to multiclass problems with a little work).

Given a (starting) classifier $G_1(\mathbf{x})$ (that outputs -1/+1), we generate a new classifier weighted toward those points that $G_1(\mathbf{x})$ misclassified, and then iterate. The algorithm is as follows:

- Initialize observations weights $w_i = 1/n$ for $i = 1, \dots, n$
- For each $m = 1, \dots, M$,
  - Fit classifier $\hat{G}_m(\mathbf{x})$ to the training data using weights $\{w_i\}$.
  - Compute
$$\text{err}_m = \frac{\sum_{i=1}^n w_i \mathbb{1}_{[y_i \neq G_m(\mathbf{x}_i)]}}{\sum_{i=1}^n w_i}.$$
  - Compute
$$\alpha_m = \log\left(\frac{1 - \text{err}_m}{\text{err}_m}\right).$$

  – Set
$$w_i \leftarrow w_i e^{\alpha_m \mathbb{1}_{[y_i \neq G_m(\mathbf{x}_i)]}}, \quad i = 1, \dots, n$$

- The boosted model is
$$\hat{G}(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(\mathbf{x})\right).$$

Note that in the loop we have to weight the observations – this can be done by bootstrapping the dataset where $(\mathbf{x}_i, y_i)$ gets weight $w_i/(\sum_{j=1}^{n} w_j)$.

Example 14. *Suppose $n = 5$, so we start with weights $w_i = 1/5$ for $i = 1, \dots, 5$. At step 1:*

- *Fit $G_1(\mathbf{x})$ and suppose y_1 and y_2 are misclassified while y_3, y_4 and y_5 are correctly classified.*
- *err_1 would be 2/5 and $\alpha_1 = \log(3/2)$.*
- *$w_1 = w_2 = \frac{1}{5}e^{\log(3/2)} = 3/10$ and $w_3 = w_4 = w_5 = \frac{1}{5}$ (note $w_1, w_2 > w_3, w_4, w_5$).*
- *Resample data according to $w_i/(\sum_{j=1}^{5} w_j)$ and fit the next classifier $G_2(\mathbf{x})$.*

In the algorithm the last step suggests that we will tend to favor y_1 and y_2 in the next step – these are the misclassified points, so the next $G_2(\mathbf{x})$ will work harder to fix these classifications.

The inspiration here is to use *easy* classifiers for your Gs, classifiers that don't take much to estimate and probably won't do well by themselves, but let each subsequent one fix the prior one's problems. For example, a tree stump (a tree with only one split) is easy and quick to fit, but by itself won't do a good job. AdaBoost was developed in the 90s and is still talked about today, so boosting in general is a very good idea!

9.6.2 Boosted regression trees

Regression trees are for the case when Y is quantitative/continuous. The basic boosting algorithm for regression trees is as follows:

- Set $\hat{f}(\mathbf{x}) = 0$ and $r_i = y_i$ for all $i = 1, \dots, n$
- For each $b = 1, \dots, B$,
  - Fit tree $\hat{f}^b$ with d splits (so $d+1$ terminal nodes) to training data $(\mathbf{X}, \mathbf{r})$
  - Update $\hat{f}$ by adding shrunken version of new tree,
  $$\hat{f}(\mathbf{x}) \leftarrow \hat{f}(\mathbf{x}) + \lambda \hat{f}^b(\mathbf{x})$$
  - Update the residuals
  $$r_i \leftarrow r_i - \lambda \hat{f}^b(\mathbf{x}_i).$$
- The boosted model is
$$\hat{f}(\mathbf{x}) = \sum_{b=1}^{B} \lambda \hat{f}^b(\mathbf{x}).$$

Instead of *fitting the data hard* using a single tree that is grown deep, the boosting approach *learns slowly*. The trees slowly improve the model fit in locations where the prior model fit performs poorly. The mantra in much of statistical learning is that model that learn slowly tend to perform well.

There are three tuning parameters in boosting:

- The number of trees B. Boosting can overfit if B is too large, so we typically use cross-validation to identify it.
- The shrinkage parameter λ controls the rate at which learning occurs. Small values like 0.001 or 0.01 are typical.
- The number d of splits in each tree. Often $d = 1$ works well, in which case we call the tree a *stump* which has a single split. d is also called the interaction depth.

This is just the tip of the iceberg for tree-based methods, there are many generalizations that lead to some of the most competitive statistical learning methods out there, and this is a very active area of current research.

[R example (Boosting.R)]

10 Neural Networks and Deep Learning

So far we have seen many models that essentially take on the structure $Y = f(\mathbf{x}) + \varepsilon$ with $\mathbf{x} = (x_1, \ldots, x_p)^{\mathrm{T}}$ for various choices of $f(\mathbf{x})$. *Neural networks* have an interesting approach to specifying f as a composition of nonlinear functions applied to various regressions.

10.1 Single Layer Neural Networks

A *feed-forward single layer* neural network uses the p predictors in $\mathbf{x}$ in the *input layer*, and feeds these into K *hidden units* (where we pick K). The form of the function is

$$f(\mathbf{X}) = \beta_0 + \sum_{k=1}^{K} \beta_k g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} x_j\right)$$

using a so-called *activation function* g to produce K *activations*

$$A_k = g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} x_j\right).$$

Note these activations are a *hidden layer* in a regular regression model

$$f(\mathbf{X}) = \beta_0 + \sum_{k=1}^{K} \beta_k A_k$$

where the $A_1, \ldots, A_K$ are new predictors that we have constructed out of the original p features.

[Picture of Figure 10.1 ISLR 2021]

The activation function g is, these days, typically chosen to be a *rectified linear unit (ReLU)* function,

$$g(z) = (z)_+ = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{otherwise.} \end{cases}$$

Historically the *sigmoid* function was used,

$$g(z) = \frac{e^z}{1 + e^z} = \frac{1}{1 + e^{-z}},$$

the same as a logistic regression function.

Note that it is important that g is a nonlinear function, otherwise everything would just end up as a multiple regression model. Let's check: assume $g(z) = z$, then

$$\begin{aligned}
f(\mathbf{x}) &= \beta_0 + \sum_{k=1}^{K} \beta_k g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} x_j\right) \\
&= \beta_0 + \sum_{k=1}^{K} \beta_k \left(w_{k0} + \sum_{j=1}^{p} w_{kj} x_j\right) \\
&= \left(\beta_0 + \sum_{k=1}^{K} \beta_k w_{k0}\right) + \sum_{j=1}^{p} \left(\sum_{k=1}^{K} \beta_k w_{kj}\right) x_j \\
&= \gamma_0 + \sum_{j=1}^{p} \gamma_j x_j
\end{aligned}$$

with regression parameters $\gamma_0, \dots, \gamma_p$ defined as wacky combinations of the neural network parameters.

One cool thing about this construction is that interactive effects can appear, even though we are only using linear functions within the deepest layer. Let $p = 2$ and $K = 2$ with $g(z) = z^2$, then assume the following parameters:

$$\begin{aligned}
\beta_0 &= 0, \quad \beta_1 = \frac{1}{4}, \quad \beta_2 = -\frac{1}{4} \\
w_{10} &= 0, \quad w_{11} = 1, \quad w_{12} = 1 \\
w_{20} &= 0, \quad w_{21} = 1, \quad w_{22} = -1.
\end{aligned}$$

The model is:

$$\begin{aligned}
f(\mathbf{x}) &= \beta_0 + \sum_{k=1}^{K} \beta_k g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} x_j\right) \\
&= \beta_0 + \sum_{k=1}^{2} \beta_k (w_{k0} + w_{k1} x_1 + w_{k2} x_2)^2 \\
&= \beta_0 + \beta_1 (w_{10} + w_{11} x_1 + w_{12} x_2)^2 + \beta_2 (w_{20} + w_{21} x_1 + w_{22} x_2)^2 \\
&= \frac{1}{4}(x_1 + x_2)^2 - \frac{1}{4}(x_1 - x_2)^2 \\
&= x_1 x_2.
\end{aligned}$$

One big difficulty is that we need to estimate a lot of parameters including $\beta_0, \beta_1, \dots, \beta_K$ and $w_{10}, \dots, w_{Kp}$. For regression models this is typically done by minimizing the mean squared error

$$\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))^2.$$

We'll discuss the estimation strategy at the end of the chapter.

For classification problems suppose there are M classes of outcomes, for example $Y \in \{1, 2, \ldots, M\}$. We use the same trick as in logistic regression and set

$$P(Y = m|\mathbf{x}) = \frac{e^{Z_m}}{\sum_{\ell=1}^{M} e^{Z_\ell}}$$

using the so-called *softmax* function where

$$Z_m = f_m(\mathbf{x}) \quad \text{for } m = 1, \ldots, M$$

is the output of M different neural networks. Estimation of probabilities follows by minimizing the *cross-entropy*,

$$-\sum_{i=1}^{n}\sum_{m=1}^{M} y_{im} \log(f_m(\mathbf{x}_i)).$$

10.2 Multilayer and Deep Neural Networks

A multilayer neural network gives rise to deep networks, and *deep learning.* We "simply" iterate the idea of hidden layers. Let's build up from the prior section.

Recall the single hidden layer model:

$$\begin{aligned} f(\mathbf{x}) &= \beta_0 + \sum_{k=1}^{K} \beta_k A_k \\ A_k &= g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} x_j\right). \end{aligned}$$

For a two-hidden-layer model with K_1 nodes in the first layer and K_2 nodes in the second layer we have

$$\begin{aligned}
f(\mathbf{x}) &= \beta_0 + \sum_{k=1}^{K_2} \beta_k A_k^{(2)} \\
A_k^{(2)} &= g\left(w_{k0}^{(2)} + \sum_{j=1}^{K_1} w_{kj}^{(2)} A_j^{(1)}\right), \qquad k = 1, \ldots, K_2 \\
A_k^{(1)} &= g\left(w_{k0}^{(1)} + \sum_{j=1}^{p} w_{kj}^{(1)} x_j\right), \qquad k = 1, \ldots, K_1
\end{aligned}$$

The notation gets a bit annoying, note the w_{kj}s now have a $w_{kj}^{(\ell)}$ to tell us which layer the weights are associated with (ℓth in this example). so you can see how this would generalize to an L layer deep model,

$$\begin{aligned}
f(\mathbf{x}) &= \beta_0 + \sum_{k=1}^{K_L} \beta_k A_k^{(L)} \\
A_k^{(L)} &= g\left(w_{k0}^{(L)} + \sum_{j=1}^{K_{L-1}} w_{kj}^{(L)} A_j^{(L-1)}\right), \qquad k = 1, \ldots, K_L \\
\vdots &= \vdots \\
A_k^{(2)} &= g\left(w_{k0}^{(2)} + \sum_{j=1}^{K_1} w_{kj}^{(2)} A_j^{(1)}\right), \qquad k = 1, \ldots, K_2 \\
A_k^{(1)} &= g\left(w_{k0}^{(1)} + \sum_{j=1}^{p} w_{kj}^{(1)} x_j\right), \qquad k = 1, \ldots, K_1.
\end{aligned}$$

Since these are just linear combinations of activations at each stage (that then pass through a nonlinear function), we can organize all of the weights into matrices. For example, in the L layer model we have the first weight matrix W_1 of dimension $(p+1) \times K_1$, the second weight matrix W_2 has dimension $(K_1 + 1) \times K_2$. Note the $+1$s come from the so-called *bias* term, which should not be confused with statistical bias! The final layer has $B = (K_L + 1)$ β parameters.

[Write out a couple of example weight matrices]

Just for fun let's count parameters for a small, realistic case. In the MNIST handwritten dataset (see 10.2 of ISLR) we have 60,000 training images with $p = 28 \times 28 = 784$ pixels, used as features. In a relatively small two-layer neural network we might choose to have $K_1 = 256$ and $K_2 = 128$, this yields $(785 \times 256) = 200,960$ in the first weight matrix alone!

10.3 Estimation

Estimation of a deep neural network (or even a shallow one) is difficult due partly to the number of parameters in the model. It turns out that the problem is nonconvex, and so there are multiple solutions that minimize

$$\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))^2.$$

The main approach to fitting is through *gradient descent*. Call all the parameters in the model $\boldsymbol{\theta}$ (i.e., all the βs and ws). Then let

$$R(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - f_{\boldsymbol{\theta}}(\mathbf{x}_i))^2$$

where we make explicit that f depends on parameters of $\boldsymbol{\theta}$. The basic gradient descent algorithm works as:

1. Start with guess $\boldsymbol{\theta}^0$ for all parameters and set $t = 0$.
2. Iterate until the MSE fails to decrease:
   (a) Find a vector δ that reflects a small change in $\boldsymbol{\theta}$ such that $\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t + \delta$ reduces the MSE, i.e., $R(\boldsymbol{\theta}^{t+1}) < R(\boldsymbol{\theta}^t)$.
   (b) Set $t \leftarrow t + 1$.

The key is to use a scaled version of the gradient of $R(\boldsymbol{\theta})$ for δ,

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \rho \nabla R(\boldsymbol{\theta}^t)$$

where $\nabla R(\boldsymbol{\theta}^t)$ is the gradient of R evaluated at $\boldsymbol{\theta}^t$. The parameter ρ is called the *learning rate* and is typically chosen to be small so that we learn slowly.

How is the gradient calculated? Via the chain rule, which ends up being, apart from an annoying bookkeeping exercise, straightforward for neural networks. Let's look for a single layer feed forward network:

$$\begin{aligned}
R_i(\boldsymbol{\theta}) &= \frac{1}{2}(y_i - f_{\boldsymbol{\theta}}(\mathbf{x}_i))^2 \\
&= \frac{1}{2}\left(y_i - \beta_0 - \sum_{k=1}^{K} \beta_k g\left(w_{k0} + \sum_{j=1}^{p} w_{kj}x_{ij}\right)\right)^2 \\
&= \frac{1}{2}\left(y_i - \beta_0 - \sum_{k=1}^{K} \beta_k g(z_{ik})\right)^2
\end{aligned}$$

where we introduce, for ease of notation, the variable z_{ik}.

Using the chain rule we have:

$$\begin{aligned}
\frac{\partial R_i(\boldsymbol{\theta})}{\partial \beta_k} &= \frac{\partial R_i(\boldsymbol{\theta})}{\partial f_{\boldsymbol{\theta}}(\mathbf{x}_i)} \cdot \frac{\partial f_{\boldsymbol{\theta}}(\mathbf{x}_i)}{\partial \beta_k} \\
&= -(y_i - f_{\boldsymbol{\theta}}(\mathbf{x}_i)) \cdot g(z_{ik}).
\end{aligned}$$

and

$$\begin{aligned}
\frac{\partial R_i(\boldsymbol{\theta})}{\partial w_{kj}} &= \frac{\partial R_i(\boldsymbol{\theta})}{\partial f_{\boldsymbol{\theta}}(\mathbf{x}_i)} \cdot \frac{\partial f_{\boldsymbol{\theta}}(\mathbf{x}_i)}{\partial g(z_{ik})} \cdot \frac{\partial g(z_{ik})}{\partial z_{ik}} \cdot \frac{\partial z_{ik}}{\partial w_{kj}} \\
&= -(y_i - f_{\boldsymbol{\theta}}(\mathbf{x}_i)) \cdot \beta_k \cdot g'(z_{ik}) \cdot x_{ij}.
\end{aligned}$$

Both of these expressions contain the residual $(y_i - f_{\boldsymbol{\theta}}(\mathbf{x}_i))$, so differentiation assigns a fraction of the residual to the parameters, this is known as *backpropagation.*

It turns out that we still need a few other tricks up our sleeve because of the large number of parameters and data points.

- For a sample of size n, randomly choosing a smaller number at each gradient step is called a *minibatch*; this process is known as *stochastic gradient descent* and is the state of the art for learning deep neural nets.
- Regularization is also often useful, adding in a ridge regression or lasso penalty for the parameters.

- *Dropout* learning randomly removes a fraction of the units in a layer when fitting; this is done randomly each time a training observation is processed. This prevents nodes from becoming over-specialized and is another form of regularization.

[R example (NeuralNetworks.R)]

[Python example (NeuralNetworksTensorFlow.ipynb)]

10.4 Convolutional Neural Networks

Convolutional neural networks (CNNs) represent the state-of-the-art method for image classification, and rely heavily on the spatially-structured nature of images. The basic idea is to ingest an image as a matrix (rather than just a vector of perhaps unrelated features) into the first layer, do a spatial convolution of the image, reduce the dimension, rinse and repeat a few times; a final layer or two are added to transfer the dimension to the number of classes.

To start with, think of an image as just visualizing a matrix of values – a grayscale image has pixels with values that are (usually) $0, 1, 2, \ldots, 255$ with 0 representing black and 255 representing white, and everything in between different shades of gray. The key is that nearby pixel values are probably related to one another, because they are "geographically" close, and a convolution tries to exploit this fact.

10.4.1 The Convolution Part of CNNs

A convolution is the application of a spatial filter to an image by sweeping it over the image in rows; this can be thought of as scrubbing a paintbrush over the image from left to right starting at the top, then going down a little, scrubbing back to the left, go down, back to the right, etc. until you reach the bottom corner. In this silly example, a paintbrush would smear out the image locally, producing a blurry version of the original image. It turns out we can define clever new kinds of paintbrushes that will extract all sorts of interesting and useful pieces of the image that help a standard neural network architecture learn a classification rule. (Note this simile is actually not quite correct: the way a real convolution works is that as we swipe from one side to the other, we pick up the paintbrush and smush it down in a spot, then get a brand new original image, move to the right a little, smush the paintbrush, get a brand new image, move to the right, smush, etc., the last step is to take all of the smushed pieces and paste them together).

Let's define the simplest kind of convolution filter (and the most commonly used in practice): the 3×3 filter. This is best done via example. Define a "data" matrix

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix}$$

and a convolution

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

note this is a silly convolution, just for illustration purposes. We start by going to the upper 3×3 block of the data matrix, multiply entrywise by the convolution weights, and then just add up; this gives us the $(1, 1)$ entry of our new matrix; then shift to the right by one and repeat (i.e., use the first three rows and columns 2-4 of the data matrix) to get the $(1, 2)$ entry of the new matrix. The convolution of the data matrix with this filter would yield

$$\begin{pmatrix} f+g+k & g+h+l \\ j+k+o & k+l+p \end{pmatrix}$$

Note we lose two rows/columns compared to the original image (we went from 4×4 to 2×2) because the convolution can't go "beyond" the bounds of the matrix. There are ways around this, you can pad the data matrix with zeroes on all sides, or you can pad the original matrix by (reverse) repeated entries of the original matrix (a period embedding of the original matrix).

Other convolutions can be imagined:

$$\begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}$$

would smooth the data because we locally-average the nearby pixels

$$\begin{pmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

would act like a first-order derivative in the x-direction

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{pmatrix}$$

would act like a first-order derivative in the y-direction.

The spatial convolution has (at least) two nice properties: 1) it extracts local features of the image and 2) it (slightly) reduces the dimension of the image. At this step we then apply the activation function to the convolved matrix to each element of the matrix separately.

Often it is helpful to further reduce the size of the (activated) convolved image through a further step called max-pooling. Max-pooling is just taking the max of a subset of the matrix, so, e.g., a 2×2 max-pooling operation on

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 6 & 3 & 4 & 7 \\ 10 & 5 & 8 & 2 \\ 9 & 8 & 11 & 2 \end{pmatrix}$$

would yield

$$\begin{pmatrix} 6 & 7 \\ 10 & 11 \end{pmatrix}$$

by taking the maximal entry of each 2×2 submatrix. Note this is a bit different than doing a regular convolution (where the operation is taking a maximum rather than a weighted combination of the 4 values) because we don't "slide" the 2×2 over, we pick it up and put it back down. This is called a *stride* in the literature, and other strides can be used, e.g. skipping a few cells and then pooling (this could be useful to bring values of the convolved image that are spatially far apart closer together). Max pooling also provides some *location invariance* in that it doesn't matter whether the biggest value happened in the upper-left, upper-right, lower-left or right piece of the 2×2 submatrix, it gets pulled out.

Example 15. *If we start with a matrix of dimension* 32×32 *and do a* 3×3 *convolutional layer, followed by a* 2×2 *max-pooling step, we would have matrices that go in dimensions:*

$$32 \times 32 \to 30 \times 30 \to 15 \times 15$$

Note that max-pooling halves the size of the matrix; if the original matrix is odd then different choices can be made (e.g., pad again, ignore the last row/column, repeat the last row/column, etc.).

10.4.2 The Neural Network Part of CNNs

Before connecting the convolution idea through to CNNs, we note that most images are not grayscale, but rather color. In color images, each pixel is represented as a (red,green,blue) (or RGB for short) triplet of values (that typically run in integers from 0 to 255). We would say that a single image (regardless of dimension) has 3 *channels*, and each channel is a matrix (in other words we can take a picture of a cat and decompose it into three separate pictures of red-, green- and blue-intensities). A convolution must act over each channel separately, so 3 input channel matrices each receive a (potentially different) convolution, and they are then added together to result in a single output matrix. This output matrix then is activated using ReLU on each element separately, followed by a max-pooling step.

The key to CNNs is that the first convolution step is actually done *many* times using different convolutions. Suppose we want K hidden nodes in the first layer – for RGB images we need $3K$ convolutional filters. At this first layer, we would say that we have K channels which are just new matrices constructed from the original convolved RGB combinations; thus we have gone from 3 to K channels. Before pooling, we apply an activation function at each pixel over all channels (this step is sometimes separated out from the convolution and called a *detector* layer).

These convolution, activation and pooling operations are then layered together to form a deep CNN. The last stage of the deep CNN is to take the last set of pooled matrices, vectorize them into a single long vector (a process called *flattening*) and then do a final layer to the desired output dimension.

(Picture ala Figure 10.8 in ISLR)

Example 16. *Suppose we have a bunch of images of dimension 32×32 which are color images with three channels (RGB) that fall into one of 10 classes. Let's figure out the dimensions and parameters of the calculations involved if we do the following "deep" CNN:*

- *Input arrays of dimension (32,32,3) for image size (32×32) and channel (3 for RGB)*
- *Do a 3×3 convolutional layer 32 times (i.e., make 32 new channels); this outputs an array of dimension (30,30,32) where the 30s come from losing the edges due to*

the convolution; the last 32 is the number of new channels. [Parameters in layer = (3 input channels) × (9 parameters per convolution) × (32 convolutions) + (32 bias parameters) = 896]. Note the RGB channels are added together at this stage.

- *ReLU previous step, no change*
- *Do a 2 × 2 max-pooling, giving array (15,15,32)*
- *Do another 3 × 3 convolutional layer 64 times, outputs (13,13,64). [Parameters in this layer = (32 input channels) × (9 parameters per convolution) × (64 convolutions) + (64 biases) = 18496].*
- *ReLU previous step, no change*
- *Do a 2 × 2 max-pooling, giving array (6,6,64)*
- *Final 3 × 3 convolutional layer with 64 channels, outputs (4,4,64) [Parameters in this layer = (64 input channels) × (9 parameters per convolution) × (64 convolutions) + (64 biases) = 36928].*
- *"Flatten" these numbers into a vector of length 4 × 4 × 64 = 1024*
- *Create a dense layer with 64 nodes, outputs vector of length 64 [Parameters in this layer = (1024 input features) × (64 output features) + (64 biases) = 65600].*
- *ReLU previous step, no change*
- *Final dense layer with 10 nodes corresponding to the image classes. [Parameters in this layer = (64 input features) × (10 output classes) + (10 biases) = 650].*

The total number of parameters is 122570, whew!

Note that the number of original images (i.e., the number of data points!) doesn't enter these calculations.

Thought experiment: what are the important local features of an image that we would want to extract (i.e., that might be useful in classifying an image), and how would we encode these into convolutional filters? Answer: put on your machine learner's hat and don't worry about it, let the weights be learned.

Fitting a deep CNN uses the same principles as fitting a regular neural network, we are just using the structure of the data to build weights in a particular way.

When fitting a CNN to "natural" images, i.e., pictures of real-world things (like cats, it's always cats), it is often important to use *data augmentation*. Data augmentation takes existing pictures, and changes them so that they look different to the network, but are still pictures of the same object. For example, stretching, zooming, flipping, rotating and/or cropping an image of a cat will still look like a cat to our eyes, and should thus be classified as such – taking a single natural image we can create a whole host of "augmented" images from it to help our network understand the important features that make up that particular class.

[Python example (CNNsTensorFlow.ipynb)]

11 Principal Component Analysis

[This chapter closely follows the development of Methods of Multivariate Analysis by Rencher and Christensen (2012)]

Principal component analysis (PCA) is an *unsupervised learning* technique in that it can be applied to datasets without a response. It has many uses, but can be used in the supervised case as a method to reduce the dimension of a (large) set of features.

In words, the principal components are the set of coefficients on vectors that are linear combinations of the data vectors that point in directions of decreasing maximal variance.

[Scatterplots of two 2d datasets, one nearly collinear, one correlated, think about representing the first and second using a single, or two, numbers to represent largeness – those are the pcs]

Throughout this section we suppose we have n observation vectors $\mathbf{x}_1, \dots, \mathbf{x}_n$ of length p (e.g., a set of n data points with p predictors). Moreover, we will suppose that the data have been centered, so $\mathbf{x}_i$ represents $\mathbf{x}_i - \bar{\mathbf{x}}$ for $i = 1, \dots, n$.

Warmup: Recall that if $\mathbf{M}$ is a real symmetric $p \times p$ matrix, then its spectral (or eigen-decomposition) is

$$\mathbf{M} = \mathbf{A}^{\mathrm{T}}\mathbf{D}\mathbf{A}$$

where $\mathbf{A}^{\mathrm{T}}$ is a $p \times p$ matrix whose columns are the eigenvectors of $\mathbf{M}$ and $\mathbf{D}$ is a $p \times p$ diagonal matrix containing the eigenvalues. [Note that the usual definition is $\mathbf{M} = \mathbf{A}\mathbf{D}\mathbf{A}^{\mathrm{T}}$; we are defining the decomposition with respect to the columns of $\mathbf{A}^{\mathrm{T}}$ instead of $\mathbf{A}$ to match up with PCA next]. We use the convention that the eigenvectors are normalized, so $\mathbf{A}\mathbf{A}^{\mathrm{T}} = \mathbf{I}$, i.e., $\mathbf{A}$ is an orthogonal matrix. Also, $\mathbf{A}^{\mathrm{T}}$ diagonalizes $\mathbf{M}$ in that $\mathbf{A}\mathbf{M}\mathbf{A}^{\mathrm{T}} = \mathbf{D}$.

The closely related singular value decomposition of a $n \times p$ matrix $\mathbf{M}$ (where we think of $p > n$) is

$$\mathbf{M} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathrm{T}}$$

where $\mathbf{U}$ is $n \times p$, $\mathbf{D}$ is $p \times p$ and $\mathbf{V}$ is $p \times p$. $\mathbf{D}$ is a diagonal matrix $\mathbf{D} = \mathrm{diag}(\sigma_1, \dots, \sigma_p)$ of *singular values* that are the (positive) square roots of the eigenvalues of $\mathbf{M}^{\mathrm{T}}\mathbf{M}$ (or $\mathbf{M}\mathbf{M}^{\mathrm{T}}$). The p columns of $\mathbf{U}$ are called the left singular vectors, and are the (first p) normalized eigenvectors of $\mathbf{MM}^{\mathrm{T}}$ while the p columns of $\mathbf{V}$ are called the right singular vectors, and are the normalized eigenvectors of $\mathbf{M}^{\mathrm{T}}\mathbf{M}$. [Some definitions depend on the rank of the matrix $\mathbf{M}$].

If the matrix $\mathbf{M}$ is nonnegative definite (and thus square), the SVD and eigendecomposition are the same.

Example 17. *If we have three vectors* $\mathbf{x}_1 = (-10, 1)^{\mathrm{T}}, \mathbf{x}_2 = (12, -0.5)^{\mathrm{T}}$ *and* $\mathbf{x}_3 = (2, 0.5)^{\mathrm{T}}$ *then we can write*

$$\begin{aligned} \mathbf{x}_1 &= -10\mathbf{e}_1 + 1\mathbf{e}_2 \\ \mathbf{x}_2 &= 12\mathbf{e}_1 - 0.5\mathbf{e}_2 \\ \mathbf{x}_3 &= -2\mathbf{e}_1 + 0.5\mathbf{e}_2 \end{aligned}$$

where $\mathbf{e}_1$ *and* $\mathbf{e}_2$ *are the standard unit vectors. Now note that the* x*-direction number is quite a bit larger than the* y*-direction ones, so we could basically refer to* $\mathbf{x}_1$ *as* -10*,* $\mathbf{x}_2$ *as* 12 *and* $\mathbf{x}_3$ *as* -2*, effectively ignoring the* y*-value.*

In this example, the variation in the $\mathbf{e}_1$ direction is the strongest. The basic idea behind PCA is to take a vector like

$$\mathbf{x} = a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + \cdots a_p\mathbf{e}_p$$

and re-express it as

$$\mathbf{x} = z_1\mathbf{a}_1 + z_2\mathbf{a}_2 + \cdots z_p\mathbf{a}_p$$

where z_1 is the biggest, z_2 the second biggest, ..., z_p is the smallest, i.e., figuring out a new coordinate system (or basis) such that we can organize coefficients into the "most important" to "least important". The numbers $z_1, \ldots, z_p$ are the principal components.

11.1 Geometric/Probabilistic Approach

As a connector, it is useful to consider the probabilistic version of one use of an eigendecomposition.

Example 18. *Suppose* $\mathbf{x}$ *is a random vector with covariance matrix* Σ. *We desire an orthogonal matrix* $\mathbf{A}$ *(recall orthogonal means* $\mathbf{A}^{\mathrm{T}}\mathbf{A} = \mathbf{I}$*) that decorrelates* $\mathbf{x}$*:* $\mathrm{Var}(\mathbf{A}\mathbf{x}) = \mathbf{D}$ *where* $\mathbf{D}$ *is a diagonal matrix. But note*

$$\mathrm{Var}(\mathbf{A}\mathbf{x}) = \mathbf{A}\Sigma\mathbf{A}^{\mathrm{T}} = \mathbf{D}$$

multiplying on both sides by $\mathbf{A}^{\mathrm{T}}$ *and* $\mathbf{A}$ *respectively yields*

$$\Sigma = \mathbf{A}^{\mathrm{T}}\mathbf{D}\mathbf{A}.$$

Since Σ *is positive definite (and thus real and symmetric), this looks a lot like an eigendecomposition of* Σ*!*

Thus, we can define the rows of $\mathbf{A}$ *to be just the eigenvectors of* Σ*, which then implies* $\mathbf{D}$ *is diagonal with the corresponding eigenvalues of* Σ*.*

Now consider the finite-sample version with n data points of p features/covariates/variables, $\mathbf{x}_1, \ldots, \mathbf{x}_n$. One way to frame our goal is that we would like to transform, for instance,

$$\mathbf{x}_6 = \begin{pmatrix} x_{61} \\ x_{62} \\ \vdots \\ x_{6p} \end{pmatrix} \to \begin{pmatrix} z_{61} \\ z_{62} \\ \vdots \\ z_{6p} \end{pmatrix} = \mathbf{z}_6$$

into "new features" that are uncorrelated. But, importantly, we want to do it in such a way that z_{61} is "more important" or "bigger" than $z_{62}, \ldots, z_{6p}$, and z_{62} is the "next biggest" and, ..., z_{6p} is the least important.

We need a convention for storing the data. We will set $\mathbf{X}^{\mathrm{T}} = [\mathbf{x}_1\, \mathbf{x}_2 \cdots \mathbf{x}_n]$, or equivalently

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1^{\mathrm{T}} \\ \mathbf{x}_2^{\mathrm{T}} \\ \vdots \\ \mathbf{x}_n^{\mathrm{T}} \end{pmatrix}.$$

Now, if $\mathbf{X}$ the $n \times p$ matrix whose *rows* are individual data vectors, then we define the $p \times p$ sample covariance matrix as

$$\hat{\Sigma} = \frac{1}{n-1}\mathbf{X}^{\mathrm{T}}\mathbf{X},$$

and note that this is a covariance matrix of "old features". Mathematically, we can cast our goal as seeking a $p \times p$ matrix $\mathbf{A}$ such that $\mathbf{z}_i = \mathbf{A}\mathbf{x}_i, i = 1, \ldots, n$ have elements that are decorrelated. Importantly, we are *not* saying $\mathbf{z}_1$ and $\mathbf{z}_2$ are uncorrelated – we are saying the variables in $\mathbf{z}_i = (z_{i1}, z_{i2}, \ldots, z_{ip})^{\mathrm{T}}$ are like p uncorrelated random variables.

Define $\mathbf{Z}^{\mathrm{T}}$ in a similar way to $\mathbf{X}^{\mathrm{T}}$, and note this transformation of every $\mathbf{x}_i$ can be summarized in matrix notation as

$$\mathbf{Z}^{\mathrm{T}} = [\mathbf{z}_1 \, \mathbf{z}_2 \cdots \mathbf{z}_n] = [\mathbf{A}\mathbf{x}_1 \, \mathbf{A}\mathbf{x}_2 \cdots \mathbf{A}\mathbf{x}_n] = \mathbf{A}\mathbf{X}^{\mathrm{T}}$$

where

$$\mathbf{Z} = \begin{pmatrix} z_{11} & z_{12} & \cdots & z_{1p} \\ z_{21} & z_{22} & \cdots & z_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ z_{n1} & z_{n2} & \cdots & z_{np} \end{pmatrix}.$$

Said equivalently, we want to find a $\mathbf{A}$ such that the (sample) covariance matrix of $\mathbf{Z}$ is diagonalized:

$$\hat{\Sigma}_z = \frac{1}{n-1}\mathbf{Z}^{\mathrm{T}}\mathbf{Z} = \frac{1}{n-1}\mathbf{A}(\mathbf{X}^{\mathrm{T}}\mathbf{X})\mathbf{A}^{\mathrm{T}} = \mathbf{A}\hat{\Sigma}\mathbf{A}^{\mathrm{T}}.$$

According to the same argument as the generic random vector case, we want to set the rows of $\mathbf{A}$ to be the eigenvectors of $\hat{\Sigma}$. In this case,

$$\hat{\Sigma} = \mathbf{A}^{\mathrm{T}}\mathbf{D}\mathbf{A}$$

and

$$\hat{\Sigma}_z = \mathbf{A}\mathbf{A}^{\mathrm{T}}\mathbf{D}\mathbf{A}\mathbf{A}^{\mathrm{T}} = \mathbf{D} = \mathrm{diag}(\lambda_1, \ldots, \lambda_p)$$

where $\{\lambda_i\}$ are eigenvalues of $\hat{\Sigma}_z$. In particular, λ_1 is the sample variance of "new feature 1": $z_{11}, z_{21}, \ldots, z_{n1}$,

$$\lambda_1 = \frac{1}{n-1}\sum_{i=1}^{n} z_{i1}^2,$$

and generally, λ_j is the sample variance of "new feature j": $z_{1j}, z_{2j}, \ldots, z_{nj}$,

$$\lambda_j = \frac{1}{n-1}\sum_{i=1}^{n}(z_{ij} - \overline{z_{\cdot j}})^2.$$

Or, in other words, λ_j is the sample variance of the jth column of $\mathbf{Z}$.

What is the geometric interpretation of this transformation? Well, $\mathbf{A}$ is an orthogonal matrix, so the transformation $\mathbf{A}\mathbf{x}_i = \mathbf{z}_i$ doesn't affect the length of the vector:

$$\|\mathbf{z}_i\|^2 = \mathbf{z}_i^{\mathrm{T}}\mathbf{z}_i = \mathbf{x}_i^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{x}_i = \mathbf{x}_i^{\mathrm{T}}\mathbf{x}_i = \|\mathbf{x}_i\|^2.$$

Moreover, $\mathbf{z}_i$ is just $\mathbf{x}_i$ expressed with respect to the basis defined by the rows of $\mathbf{A}$, e.g., the eigenvectors of $\hat{\Sigma}$. In particular, since $\mathbf{A}^{\mathrm{T}} = [\mathbf{a}_1\,\mathbf{a}_2 \cdots \mathbf{a}_p]$ has columns that are the eigenvectors of $\hat{\Sigma}$, then

$$\mathbf{x}_i = z_{i1}\mathbf{a}_1 + z_{i2}\mathbf{a}_2 + \cdots + z_{ip}\mathbf{a}_p.$$

The coefficients in this expansion, i.e., the values in

$$\mathbf{z}_i = \begin{pmatrix} z_{i1} \\ z_{i2} \\ \vdots \\ z_{ip} \end{pmatrix}$$

are called the *principal components, scores, component scores* of data vector $\mathbf{x}_i$.

The (length p) vectors $\mathbf{a}_1, \ldots, \mathbf{a}_p$ are orthogonal, length 1, and are ordered in such a way as to point in directions of decreasing variability, since the eigenvalues are ordered with $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0$.

The 1st column of $\mathbf{Z}$ contains the first principal component scores for all data points, which are the data values expressed in the new basis. As we move out further in the columns of $\mathbf{Z}$, the column vectors get "smaller", since their variability is less according to decreasing eigenvalues, and so z_{i1} will usually be bigger than $z_{i2}, \ldots, z_{ip}$.

Another way this is often described is that the data are effectively living on a lower-dimensional space. [Picture of (X_1, X_2) with collinearity, near collinearity and orthogonality: one number can adequately represent the first two sets, but we need two numbers to represent the third]. For example, if $p = 3$ and $(\lambda_1, \lambda_2, \lambda_3) = (10.3, 8.5, 0.02)$ then the point cloud of data in three dimensions would be an elliptical pancake. Thus, the data may be adequately described on a plane, rather than in three dimensions.

As the eigenvalues are sample variances, we can define the proportion of total variance explained by the first k components as

$$\text{Proportion of variance of first } k \text{ components} = \frac{\lambda_1 + \cdots + \lambda_k}{\lambda_1 + \cdots + \lambda_p}.$$

The denominator is indeed the total variance of $\mathbf{X}$ in that

$$\operatorname{tr}(\hat{\Sigma}_z) = \lambda_1 + \cdots + \lambda_p = \operatorname{tr}(\hat{\Sigma}_z \mathbf{A}^{\mathrm{T}} \mathbf{A}) = \operatorname{tr}(\mathbf{A}^{\mathrm{T}} \hat{\Sigma}_z \mathbf{A}) = \operatorname{tr}(\hat{\Sigma}).$$

This is particular useful in practice if only a few components $k \ll p$ describe most of the variance of $\mathbf{X}$. In that case we have

$$\begin{aligned} \mathbf{x}_i &= z_{i1}\mathbf{a}_1 + z_{i2}\mathbf{a}_2 + \cdots + z_{ik}\mathbf{a}_k + \cdots + z_{ip}\mathbf{a}_p \\ &\approx z_{i1}\mathbf{a}_1 + z_{i2}\mathbf{a}_2 + \cdots + z_{ik}\mathbf{a}_k. \end{aligned}$$

Instead of using $\mathbf{x}_1, \ldots, \mathbf{x}_n$, we would instead use the reduced component scores $\mathbf{z}'_1, \ldots, \mathbf{z}'_n$ where

$$\mathbf{z}'_i = \begin{pmatrix} z_{i1} \\ z_{i2} \\ \vdots \\ z_{ik} \end{pmatrix}.$$

The vectors $\mathbf{a}_1, \ldots, \mathbf{a}_p$ are *eigenvectors*, and point in directions of (decreasing) variability. Since the eigenvectors are normalized, we can restore the magnitude of variation by multiplying them by the eigenvalues: the vectors $\lambda_1\mathbf{a}_1, \ldots, \lambda_p\mathbf{a}_p$ are called the *loadings*, and are *not* normalized. Loadings are useful to get a sense for which eigenvectors are the most important in explaining total variability. This is a point of confusion in many software packages, so check the output of any code to ensure that "loading" or "eigenvector" is referring to the correct vector (normalized vs. unnormalized).

One of the most useful properties of PCA is that it works if $n > p$ and also if $p > n$. In particular, if there are more predictors than data points, then $\hat{\Sigma}$ is singular and has eigenvalues that are exactly zero.

What is the information in the last few principal components? Suppose $\mathbf{x} = (x_1, x_2, x_3, x_4)$ and $x_4 = \sum_{j=1}^{3} x_i/3$ so that the fourth covariate is strictly a linear combination of the first three. Then $\lambda_4 = 0$ and the principal components that are coefficients of $\mathbf{a}_4 = (a_1, a_2, a_3, a_4)^{\mathrm{T}}$

have zero variance (because $\lambda_4 = 0$) – and so they are all exactly zero. Thus,

$$\begin{aligned} z_4 &= \mathbf{a}_4^{\mathrm{T}}\mathbf{x} \\ &= a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4 \\ &= a_1x_1 + a_2x_2 + a_3x_3 + a_4\sum_{j=1}^{3} x_i/3 \\ &= 0, \end{aligned}$$

for any $\mathbf{x}$, so that $\mathbf{a}_4$ is proportional to $(1, 1, 1, -3)^{\mathrm{T}}$. Generally, if $\lambda_k = 0$ for a particular k, then the corresponding eigenvector $\mathbf{a}_k$ gives the linear combination that constrains the covariates. Or, if $\lambda_k \approx 0$, there is *near* collinearity between certain combinations of the covariates, defined by $\mathbf{a}_k$.

Finally, note the connection between the singular value decomposition and PCA: the SVD of a $n \times p$ matrix $\mathbf{X}$ is

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathrm{T}}$$

where $\mathbf{V}$ contains the (normalized) eigenvectors of $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ – these are exactly the PCA eigenvectors. The eigenvalues for PCA are

$$\lambda_i = \frac{\sigma_i^2}{n-1}$$

where σ_i is the ith singular value.

[R example (PCA.R)]

11.2 Algebraic Approach

Another way to define the first principal component vector is that we seek a vector $\mathbf{a}$ (of length p) that maximizes the sample variance of $z_1 = \mathbf{a}^{\mathrm{T}}\mathbf{x}_1, \ldots, z_n = \mathbf{a}^{\mathrm{T}}\mathbf{x}_n$. The transformation is

$$\mathbf{z}^{\mathrm{T}} = (z_1\, z_2 \cdots z_n) = \mathbf{a}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}$$

where $\mathbf{X}^{\mathrm{T}} = [\mathbf{x}_1 \cdots \mathbf{x}_n]$. The sample variance of $\mathbf{z}$ is

$$\frac{1}{n-1}\mathbf{z}^{\mathrm{T}}\mathbf{z} = \frac{1}{n-1}\mathbf{a}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{a} = \mathbf{a}^{\mathrm{T}}\hat{\Sigma}\mathbf{a}.$$

Of course, sending $\mathbf{a} \to \infty$ would maximize this, so instead we seek the vector that maximizes

$$\frac{\mathbf{a}^{\mathrm{T}}\hat{\Sigma}\mathbf{a}}{\mathbf{a}^{\mathrm{T}}\mathbf{a}}.$$

The vector that does this is exactly the eigenvector of Σ with the largest eigenvalue.

11.3 Principal Component Regression

In a regression setting with

$$Y = \beta_0 + \beta_1 X_1 + \cdots \beta_p X_p + \varepsilon,$$

where $p \gg 0$, the idea behind principal component regression is to take a PCA decomposition of the covariate matrix $\mathbf{X}$ and keep only those component scores that explain most of the variability. In other words, we heuristically want $\mathbf{X} \approx \mathbf{Z}$ where $\mathbf{X}$ is $n \times p$ and $\mathbf{Z}$ is $n \times k$ with $k \ll p$. This approximation is not elementwise, but rather in terms of the spaces spanned by the rows of $\mathbf{X}$ and $\mathbf{Z}$.

Principal component regression takes the form

$$Y = \beta_0 + \beta_1 Z_1 + \cdots \beta_k Z_k + \varepsilon$$

where $Z_1, \ldots, Z_k$ are scores for the first k principal components. The choice of how many components to retain, k, is ad hoc, but some suggestions in the literature are

- Retain enough components to account for 80% of the total variability
- Retain the components whose eigenvalues are greater than the average of all eigenvalues, $\sum_{i=1}^{p} \lambda_i / p$
- Use the *scree graph*, a plot of λ_i vs. i, and look for natural breaks between the "large" and "small" eigenvalues
- Test for the significance of the "larger" components.

Thus, principal component regression can be interpreted as a type of regularization procedure (similar to ridge/lasso), but is in some sense discrete in that certain prediction vectors are entirely discarded, rather than downweighted.

One potential problem with this method is that the response does not enter the choice of how to decompose the features. The method partial least squares tries to get around this, and there have been other supervised approaches to better define principal components when there is a response.

[R example (PCRegression.R)]

12 Nonparametric Regression: Splines, Smoothing and Kernels

When the covariate X and response Y are *not* linearly related, we previously saw that we could try polynomial regression

$$Y = \beta_0 + \beta_1 X + \cdots + \beta_p X^p + \varepsilon$$

to capture nonlinear relationships. Some problems are:

- Often need p large to achieve reasonable model fits
- Extrapolated values behave extremely badly (fit is good locally, but not globally)

The basic idea of this chapter is to consider models of the form

$$Y = f(X) + \varepsilon = \sum_{j=0}^{p} \beta_j K_j(X) + \varepsilon$$

for some choices of functions $K_1, \ldots, K_p$. If

$$K_i(x) = X^i$$

this just reduces to polynomial regression. Choices include

- Step functions (breaking up the domain into chunks and fitting separate models within chunks)
- Regression and penalized splines (forcing some natural behavior at the boundary between regions)
- Smoothing splines (adding in a functional penalty)
- Local regression (locally varying linear models)
- Generalized additive models (extending these methods for multiple predictors)

12.1 Regression Splines

Our goal is to specify a flexible structure for $f(X)$ by representing it as a sum of some functions.

The *piecewise constant* model uses a step function,

$$K(x) = \mathbb{1}_{[a,b)}(x) = \begin{cases} 1 & x \in [a,b) \\ 0 & x \notin [a,b). \end{cases}$$

If the data are defined on $X \in [0,1]$ [which is not restrictive since X can always be scaled to this domain], define a set of p *knots* $0 < a_1 < a_2 < \cdots < a_p < 1$ with p corresponding functions

$$\begin{aligned} K_1(x) &= \mathbb{1}_{[a_1,a_2)}(x) \\ K_2(x) &= \mathbb{1}_{[a_2,a_3)}(x) \\ &\vdots \\ K_{p-1}(x) &= \mathbb{1}_{[a_{p-1},a_p)}(x) \\ K_p(x) &= \mathbb{1}_{[a_p,1]}(x). \end{aligned}$$

The corresponding model is

$$Y = f(X) + \varepsilon = \beta_0 + \sum_{j=1}^{p} \beta_j K_j(X) + \varepsilon.$$

Example 19. *Suppose $a_1 = 1/4$ and $a_2 = 1/2$, then*

$$f(x) = \beta_0 + \beta_1 \mathbb{1}_{[1/4,1/2)}(x) + \beta_2 \mathbb{1}_{[1/2,1]}(x) = \begin{cases} \beta_0 & x \in [0,1/4) \\ \beta_0 + \beta_1 & x \in [1/4,1/2) \\ \beta_0 + \beta_2 & x \in [1/2,1] \end{cases}$$

[Picture]

Note that $f(X)$ *only* takes on values $\beta_0, \beta_0 + \beta_1, \ldots, \beta_0 + \beta_p$ since the K_i's have disjoint supports.

If data $(x_1, y_1), \dots, (x_n, y_n)$ are available, then we can use OLS technology with the design matrix

$$\mathbf{X} = \begin{pmatrix} 1 & K_1(x_1) & K_2(x_1) & \cdots & K_p(x_1) \\ 1 & K_1(x_2) & K_2(x_2) & \cdots & K_p(x_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & K_1(x_n) & K_2(x_n) & \cdots & K_p(x_n) \end{pmatrix},$$

to estimate $\beta_0, \dots, \beta_p$ (and we get all of the usual properties of OLS estimators, confidence intervals, predictions, hypothesis testing, etc.). This results in a *piecewise constant* fit, but there are (always) breaks in $\hat{f}(x)$ at the knot boundaries a_i.

A *piecewise linear* function is *linear* between knots. For example, if $p = 1$ and $a_1 = 1/2$ with

$$K_1(x) = x\mathbb{1}_{[0,1/2)}(x)$$
$$K_2(x) = x\mathbb{1}_{[1/2,1]}(x)$$

then

$$f(x) = \beta_0 + \beta_1 K_1(x) + \beta_2 K_2(x) = \begin{cases} \beta_0 + \beta_1 x & x < 1/2 \\ \beta_0 + \beta_2 x & x \geq 1/2 \end{cases}$$

is linear on $[0, 1/2)$ and on $[1/2, 1]$. However,

$$\lim_{x \to 1/2^-} f(x) = \beta_0 + \beta_1/2 \neq \beta_0 + \beta_2/2 = \lim_{x \to 1/2^+} f(x)$$

is disjoint at the knot.

A *truncated linear spline* overcomes this by forcing continuity at the knots, for example using basis functions of the form

$$(x - a)_+ = \begin{cases} x - a & x - a > 0 \\ 0 & x - a \leq 0 \end{cases}$$

Define

$$f(x) = \beta_0 + \beta_1 x + \beta_2 (x - a)_+,$$

and note

$$f(x) = \begin{cases} \beta_0 + \beta_1 x & x < a \\ (\beta_0 - \beta_2 a) + (\beta_1 + \beta_2) x & x \geq a. \end{cases}$$

is linear on $x < a$ and $x \geq a$, and moreover $\lim_{x \to a^+} f(x) = \beta_0 + \beta_1 a = \lim_{x \to a^-} f(x)$ is continuous at the knot.

The general truncated linear spline model uses

$$K_i(x) = (x - a_i)_+$$

for knots $0 < a_1 < a_2 < \cdots < a_p < 1$, where

$$f(x) = \beta_0 + \beta_1 x + \sum_{j=1}^{p} \beta_{1i} K_i(x)$$

and is continuous everywhere, and is just a linear function between adjacent knots.

The issue with linear splines is the obvious kinks at the knots. To overcome this, a *quadratic spline* replaces $(x - a)_+$ with $(x - a)_+^2$, so that

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \sum_{i=1}^{p} \beta_{2i} (x - a_i)_+^2$$

is *continuous* and has *continuous first derivatives everywhere.*

Typically the *cubic* spline is most common.

Definition 20. *A* cubic spline *with knots* $0 < a_1 < a_2 < \cdots < a_p < 1$ *is a cubic polynomial on each of* $(0, a_1), (a_1, a_2), \ldots, (a_p, 1)$ *and with* f, f' *and* f'' *being continuous at each* a_i*.*

A cubic spline can be represented via

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \sum_{i=1}^{p} \beta_{3i} (x - a_i)_+^3$$

and has *continuous first and second derivatives* on $[0, 1]$. The third derivative is constant between knots, but is disjoint at knot boundaries. A reason cubic splines are preferred is that it uses the lowest order of polynomial such that the eye cannot detect the knot locations. However, the truncated cubic spline can exhibit high variance at the outer range of predictors and the design matrix can behave badly (with potentially extremely large values and collinearity).

B-splines (Basis-splines) provide a well-conditioned basis for spline representations that span the same space as the truncated splines. We have the convention that a spline with polynomial degree d has order d, but some packages and authors call the order $d+1$, so be careful. Supposing the spline has p *interior knots*

$$0 < a_1 < \cdots < a_p < 1,$$

we augment the set of interior knots with boundary knots $0 = a_0 = a_{-1} = a_{-2} = \cdots$ and $1 = a_{p+1} = a_{p+2} = \cdots$. We call $B_{i,j}(x)$ the ith B-spline of order j, with the B-splines of order 0 being indicator functions

$$B_{i,0}(x) = \mathbb{1}_{[a_i, a_{i+1})}(x) \quad \text{for} \quad i = 0, 1, \ldots, p$$

and note that these form a *partition of unity* in that

$$\sum_i B_{i,1}(x) = 1.$$

Higher order B-splines are obtained by recurrence with

$$B_{i,j}(x) = \frac{x - a_i}{a_{i+j} - a_i} B_{i,j-1}(x) + \frac{a_{i+j+1} - x}{a_{i+j+1} - a_{i+1}} B_{i+1,j-1}(x).$$

[Do the thought experiment of how going from $j = 0$ to $j = 1$ adds in lines, and then $j = 1$ to $j = 2$ adds in quadratics. But also we have that these can be decomposed as polynomial terms times indicator functions, and so are compactly supported].

The number of splines is $p + d + 1$. It turns out that every B-spline also forms a partition of unity,

$$\sum_i B_{i,j}(x) = 1.$$

Note that in the recursion, the coefficient weights are just *linear* functions, thus the linear B-spline consists of piecewise linear functions, and higher order B-splines are little polynomial functions that span the same space as the truncated polynomial splines. Also note that, due to the recursion, every B-spline member is just polynomial terms times indicator functions, and is thus compactly supported (i.e., only takes on nonzero values in a compact set). This last feature makes B-splines numerically stable, and is a major reason they are used extensively in software routines.

Example 21. *Suppose we have $p = 2$ interior knots $a_1 = a < a_2 = b$. Then the B-spline of order 0 consists of $2 + 0 + 1 = 3$ functions,*

$$B_{0,0}(x) = \mathbb{1}_{[0,a)}(x)$$
$$B_{1,0}(x) = \mathbb{1}_{[a,b)}(x)$$
$$B_{2,0}(x) = \mathbb{1}_{[b,1)}(x).$$

[Picture] The B-spline of order 1 consists of $2 + 1 + 2 = 4$ functions,

$$B_{-1,1}(x) = \begin{cases} \frac{a-x}{a} & x \in [0, a) \\ 0 & x \in [a, 1) \end{cases}$$

$$B_{0,1}(x) = \begin{cases} \frac{x}{a} & x \in [0, a) \\ \frac{b-x}{b-a} & x \in [a, b) \\ 0 & x \in [b, 1) \end{cases}$$

$$B_{1,1}(x) = \begin{cases} 0 & x \in [0, a) \\ \frac{x-a}{b-a} & x \in [a, b) \\ \frac{1-x}{1-b} & x \in [b, 1) \end{cases}$$

$$B_{2,1}(x) = \begin{cases} 0 & x \in [0, b) \\ \frac{x-b}{1-b} & x \in [b, 1) \end{cases}$$

[Picture] How did we know to start at $B_{-1,1}(x)$ instead of $B_{0,1}(s)$? Well the easy answer is we checked to see which $B_{\cdot,0}(x)$ functions existed, the others are defined to be zero. Using the current conventions for indexing (i, j) we will typically have the first index running $i = -(j + 1), \ldots, (j + 1)$.

Finally, the B-spline of order 2 consists of $2 + 2 + 1 = 5$ functions, the first of which is

$$B_{-2,2}(x) = \begin{cases} \left(\frac{a-x}{a}\right)^2 & x \in [0, a) \\ 0 & x \in [a, 1). \end{cases}$$

Because B-splines span the same space as truncated splines, most software packages use B-splines when constructing cubic splines due to their numerical properties (even if the regression coefficients β_i are a bit less interpretable).

Standard errors for cubic splines near the boundary of $[0, 1]$ can behave poorly. A natural spline is a regularized version of a cubic spline.

Definition 22. *A* natural cubic spline *with knots* $0 < a_1 < \cdots < a_p < 1$ *is a cubic spline that is linear on* $(0, a_1)$ *and* $(a_p, 1)$.

How should a natural cubic spline be specified? We could use

$$f(x) = e_i(x - a_i)^3 + d_i(x - a_i)^2 + c_i(x - a_i) + b_i \quad \text{for} \quad a_i \le x < a_{i+1},$$

that is, use a cubic polynomial between each knot. Continuity conditions of f, f' and f'' then put conditions on the coefficients $\{e_i, d_i, c_i, b_i\}$. There is a more useful representation, requiring some notation.

Define

$$\mathbf{f} = (f(a_1), \ldots, f(a_p))^{\mathrm{T}} \quad \text{and} \quad \boldsymbol{\gamma} = (f''(a_2), \ldots, f''(a_{p-1}))^{\mathrm{T}},$$

noting that $f''(a_1) = f''(a_p) = 0$ by the boundary conditions. It turns out that $f(x)$ can be calculated *explicitly* at *any* $x \in [0, 1]$, and only depends on $\mathbf{f}$ and $\boldsymbol{\gamma}$. More precisely, $\mathbf{f}$ and $\boldsymbol{\gamma}$ *define* the natural cubic spline (but not all choices of $\mathbf{f}$ and $\boldsymbol{\gamma}$ will result in such a spline).

Define

$$h_i = a_{i+1} - a_i, \quad i = 1, \ldots, p - 1$$

and let $\mathbf{Q}$ be the $p \times (p - 2)$ matrix

$$\mathbf{Q} = \begin{pmatrix} \frac{1}{h_1} & 0 & 0 & \cdots & 0 & 0 \\ -\frac{1}{h_1} - \frac{1}{h_2} & \frac{1}{h_2} & 0 & \cdots & 0 & 0 \\ \frac{1}{h_2} & -\frac{1}{h_2} - \frac{1}{h_3} & \frac{1}{h_3} & \cdots & 0 & 0 \\ 0 & \frac{1}{h_3} & -\frac{1}{h_3} - \frac{1}{h_4} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -\frac{1}{h_{p-3}} - \frac{1}{h_{p-2}} & \frac{1}{h_{p-2}} \\ 0 & 0 & 0 & \cdots & \frac{1}{h_{p-2}} & -\frac{1}{h_{p-2}} - \frac{1}{h_{p-1}} \\ 0 & 0 & 0 & \cdots & 0 & \frac{1}{h_{p-1}} \end{pmatrix}.$$

Let $\mathbf{R}$ be the symmetric $(p - 2) \times (p - 2)$ matrix

$$\mathbf{R} = \begin{pmatrix} \frac{1}{3}(h_1 + h_2) & \frac{1}{6}h_2 & 0 & \cdots & 0 \\ \frac{1}{6}h_2 & \frac{1}{3}(h_2 + h_3) & \frac{1}{6}h_3 & \cdots & 0 \\ 0 & \frac{1}{6}h_3 & \frac{1}{3}(h_3 + h_2) & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{1}{3}(h_{p-2} + h_{p-1}) \end{pmatrix},$$

and note that $\mathbf{R}$ is positive definite. Finally define

$$\mathbf{K} = \mathbf{Q}\mathbf{R}^{-1}\mathbf{Q}^{\mathrm{T}}.$$

Theorem 23. *The vectors* $\mathbf{f}$ *and* $\boldsymbol{\gamma}$ *define a natural cubic spline if and only if*

$$\mathbf{Q}^{\mathrm{T}}\mathbf{f} = \mathbf{R}\boldsymbol{\gamma},$$

and moreover

$$\int_0^1 f''(x)^2 \mathrm{d}x = \boldsymbol{\gamma}^{\mathrm{T}}\mathbf{R}\boldsymbol{\gamma} = \mathbf{f}^{\mathrm{T}}\mathbf{K}\mathbf{f}.$$

This theorem will have crucial implications in the next two sections.

Somewhat amazingly, a cubic spline has an equivalent representation as

$$f(x) = \beta_0 + \beta_1 x + \sum_{i=1}^{p} \beta_{1i} k(x, a_i)$$

where

$$\begin{aligned} k(x,z) = &\big((z-1/2)^2 - 1/12\big)\big((x-1/2)^2 - 1/12\big)/4 - \\ &\big((|x-z|-1/2)^4 - (1/2)(|x-z|-1/2)^2 + 7/240\big)/24. \end{aligned}$$

This formulation is particularly nice, since the integrated squared penalty can be written

$$\int_0^1 f''(x)^2 \mathrm{d}x = \boldsymbol{\beta}^{\mathrm{T}}\mathbf{S}\boldsymbol{\beta}$$

where $\boldsymbol{\beta} = (\beta_0, \beta_1, \beta_{11}, \beta_{12}, \ldots, \beta_{1p})^{\mathrm{T}}$, the first two rows and columns of $\mathbf{S}$ are zero, and the $(i+2, j+2)$th element is $k(a_i, a_j)$. [Why is this useful? Can write down a penalized regression spline solution explicitly, as in the next section].

The last remaining issue is *knot placement*.

- Place many knots near locations where you expect the function to vary a lot
- Place knots uniformly through $[0, 1]$
- Place knots at uniform quantiles of the x covariates

- Use cross-validation (e.g., remove 10% of the data) and allow the knot numbers/locations to vary, favor model with best predictive power
- Use a forward/backward selection algorithm, say starting from a dense grid of possible knots
- Use lots of knots, and regularize their coefficients (next section).

[R example (Splines1.R)]

12.2 Penalized Regression Splines

Regression splines are only as flexible as the number of knots used, however for $p \gg 0$ the regression model has many parameters and high variability of OLS estimators can be a problem. One route is to perform *automatic knot selection*, but another is to use relatively many knots, but to *regularize* the coefficients, as in lasso/ridge regression.

For example, using a truncated power basis of degree q, with p knots, the ith set of covariates is

$$\mathbf{x}_i = \left(1, x_1, x_1^2, \ldots, x_1^q, (x_1 - a_1)_+^q, (x_1 - a_2)_+^q, \ldots, (x_1 - a_p)_+^q\right)$$

yielding design matrix

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1^{\mathrm{T}} \\ \mathbf{x}_2^{\mathrm{T}} \\ \vdots \\ \mathbf{x}_n^{\mathrm{T}} \end{pmatrix}.$$

A *penalized spline* then is the minimizer of

$$(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda \boldsymbol{\beta}^{\mathrm{T}}\mathbf{D}\boldsymbol{\beta}$$

for a penalty matrix

$$\mathbf{D} = \begin{pmatrix} \mathbf{0}_{(q+1)\times(q+1)} & \mathbf{0}_{(q+1)\times p} \\ \mathbf{0}_{p\times(q+1)} & \mathbf{I}_{p\times p} \end{pmatrix}.$$

Note that the penalty can be written

$$\boldsymbol{\beta}^{\mathrm{T}}\mathbf{D}\boldsymbol{\beta} = \sum_{i=q+2}^{q+1+p} \beta_i^2$$

and thus *does not* penalize the first q-degree polynomial. This is similar to ridge regression, but with a more complicated mean function.

If $\lambda = 0$, the solution reduces to OLS, whereas if $\lambda \to \infty$, the fitted curve nears

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \cdots + \beta_q X^q + \varepsilon,$$

that is, just a q-degree polynomial regression.

For observations $\mathbf{y}$, the minimizer is

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{D})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

and the fitted values are then

$$\hat{\mathbf{y}} = \mathbf{X}(\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{D})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}.$$

The truncated power basis behaves poor, numerically, so we should recast this solution in terms of the B-spline basis of order q. In particular, there is a square invertible matrix $\mathbf{L}_q$ such that

$$\mathbf{X}_B = \mathbf{X}\mathbf{L}_q$$

where $\mathbf{X}_B$ is the corresponding B-spline basis. Then the fitted values become

$$\begin{aligned}(\mathbf{X}_B\mathbf{L}_q^{-1})\big((\mathbf{X}_B\mathbf{L}_q^{-1})^{\mathrm{T}}(\mathbf{X}_B\mathbf{L}_q^{-1}) + \lambda\mathbf{D}\big)^{-1}(\mathbf{X}_B\mathbf{L}_q^{-1})^{\mathrm{T}}\mathbf{y} &= \mathbf{X}_B\mathbf{L}_q^{-1}\big(\mathbf{L}_q^{-\mathrm{T}}\mathbf{X}_B^{\mathrm{T}}\mathbf{X}_B\mathbf{L}_q^{-1} + \lambda\mathbf{D}\big)^{-1}\mathbf{L}_q^{-\mathrm{T}}\mathbf{X}_B^{\mathrm{T}}\mathbf{y} \\ &= \mathbf{X}_B(\mathbf{X}_B^{\mathrm{T}}\mathbf{X}_B + \lambda\mathbf{L}_q^{\mathrm{T}}\mathbf{D}\mathbf{L}_q)^{-1}\mathbf{X}_B^{\mathrm{T}}\mathbf{y}\end{aligned}$$

so that the fitted values are *equivalent* to the B-spline solution under the penalty matrix $\mathbf{L}_q^{\mathrm{T}}\mathbf{D}\mathbf{L}_q$.

Other penalties may be used. For instance, Eilers and Marx (1996) suggested a fitting procedure based on n observations and p B-splines $B_1, \ldots, B_p$ of order k

$$\sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{p} a_j B_j(x_i)\right)^2 + \lambda \sum_{j=k+1}^{p}(\Delta^k a_j)^2$$

where $\Delta a_j = a_j - a_{j-1}$. For example, if $k = 1$, we have the penalty takes the form

$$\lambda \sum_{j=2}^{p}(a_j - a_{j-1})^2.$$

In particular, note this penalty *encourages adjacent a_j's to be similar.* [There is a connection to a MRF prior for this penalty].

[R example (Splines2.R)]

12.3 Smoothing Splines

A smoothing spline approaches the fitting procedure in a different way. Recall the model can be written

$$Y = f(X) + \varepsilon,$$

so we would desire to minimize

$$\sum_{i=1}^{n} (y_i - f(x_i))^2$$

over some class of functions $\{f\}$ (up to now we have seen parametric or 'nonparametric' classes of functions for f). What if we wanted to allow f to be (almost) *any* function with two continuous derivatives? The issue is that even very smooth classes can interpolate the data by producing extremely wiggly functions, i.e., we can get $RSS = 0$.

Instead, consider

$$\sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda \int_0^1 f''(x)^2 \mathrm{d}x, \tag{9}$$

where the penalty term now penalizes the (integrated squared) curvature of the fitted function f. The function that minimizes (9) is called a *cubic smoothing spline.* In fact, it turns out to be a natural cubic spline with knots at $x_1, \ldots, x_n$, but whose coefficients are *shrunken* by a relationship with λ. Conversely, a natural cubic spline with knots at every x_i yields a cubic smoothing spline.

We show this in three steps. The first is to note that, given a set of target points $z_1, \ldots, z_n$ at points $0 < x_1 < \cdots < x_n < 1$, there is a *unique* natural cubic spline f such that $f(x_i) = z_i$ for all i. To see this, if $\mathbf{z} = (z_1, \ldots, z_n)^{\mathrm{T}}$ and $\mathbf{f} = (f(x_1), \ldots, f(x_n))^{\mathrm{T}}$, then recall that $\mathbf{f}$ is a natural cubic spline if and only if there is a vector $\boldsymbol{\gamma}$ so that

$$\mathbf{Q}^{\mathrm{T}}\mathbf{f} = \mathbf{R}\boldsymbol{\gamma}.$$

Setting $\boldsymbol{\gamma} = \mathbf{R}^{-1}\mathbf{Q}^{\mathrm{T}}\mathbf{z}$ gives the unique solution.

Theorem 24. *Suppose $n \geq 2$ and f is the (unique) natural cubic spline such that $f(x_i) = z_i$ for $0 < x_1 < \cdots < x_n < 1$. Let $\tilde{f}$ be* any *other function that is continuously twice differentiable on $[0,1]$ also satisfying $\tilde{f}(x_i) = z_i$. Then*

$$\int_0^1 f''(x)^2 \mathrm{d}x \leq \int_0^1 \tilde{f}''(x)^2 \mathrm{d}x.$$

Proof.

$$\begin{aligned}\int_0^1 \tilde{f}''^2 &= \int (f'' + (\tilde{f}'' - f''))^2 \\ &= \int f''^2 + 2\int f''(\tilde{f}'' - f'') + \int (\tilde{f}'' - f'')^2 \\ &= \int f''^2 + \int (\tilde{f}'' - f'')^2 \\ &\geq \int f''^2\end{aligned}$$

with equality only if $(\tilde{f}'' - f'')$ is linear on $[0,1]$, but $(\tilde{f}'' - f'')(x_i) = 0$ for $n \geq 2$ points, so $\tilde{f} \equiv f$. The cross product is zero using integration by parts and that f is a natural cubic spline and $f(x_i) - \tilde{f}(x_i) = 0$. □

Finally, we are ready for the main theorem.

Theorem 25. *There is a unique natural spline f that minimizes (9) which is given by*

$$\mathbf{f} = (f(x_1), \ldots, f(x_n))^{\mathrm{T}} = (\mathbf{I}_n + \lambda \mathbf{K})^{-1}\mathbf{y}.$$

Proof. Let $\tilde{f}$ be any other function (with $\int \tilde{f}'' < \infty$). Then let f be the unique natural spline interpolator for $f(x_i)$, i.e., $f(x_i) = \tilde{f}(x_i)$. Trivially,

$$\sum_{i=1}^{n} (y_i - f(x_i))^2 = \sum_{i=1}^{n} (y_i - \tilde{f}(x_i))^2$$

and moreover, by the previous theorem, $\int f''^2 \leq \int \tilde{f}''^2$. □

In other words, for any candidate function, we can always find a natural cubic spline with a smaller (or equal) value of (9).

12.4 The Smoother Matrix

Recall that the predicted values for a penalized spline are of the form

$$\hat{\mathbf{y}} = \mathbf{X}(\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{D})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} = \mathbf{W}_\lambda \mathbf{y}$$

for a *weight matrix* $\mathbf{W}_\lambda$. In particular, $\hat{\mathbf{y}}$ is a *linear smoother* in that it is *linear in the observations*. Sometimes $\mathbf{W}_\lambda$ is called the *smoother matrix*.

Suppose the smoother matrix $\mathbf{W}_\lambda$ is $n \times n$ and nonnegative definite. Then let $\lambda_1 \geq \cdots \geq \lambda_n \geq 0$ be the ordered eigenvalues with associated eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$. Since the eigenvectors form a basis of $\mathbb{R}^n$, we have

$$\mathbf{y} = \sum_{i=1}^{n} \alpha_i \mathbf{v}_i$$

for some coefficients $\alpha_1, \ldots, \alpha_n$, and

$$\hat{\mathbf{y}} = \mathbf{W}_\lambda \mathbf{y} = \mathbf{W}_\lambda \sum_{i=1}^{n} \alpha_i \mathbf{v}_i = \sum_{i=1}^{n} \alpha_i \mathbf{W}_\lambda \mathbf{v}_i = \sum_{i=1}^{n} \alpha_i \lambda_i \mathbf{v}_i$$

so when $\lambda_i \approx 0$, we are discarding the effects of all eigenvectors $j \geq i$.

Recall that the trace of the hat matrix defined the degrees of freedom of a parametric model. In the same way, we define $\mathrm{tr}(\mathbf{W}_\lambda)$ as the *effective number of parameters* for a spline. A smoother with ν degrees of freedom summarizes the data about to the same extent that a $(\nu - 1)$-degree polynomial does.

For a penalized spline with N knots and degree p, we have

$$\mathrm{tr}(\mathbf{W}_0) = p + 1 + N$$

and at the other extreme, as $\lambda \to \infty$,

$$\mathrm{tr}(\mathbf{W}_\lambda) \to p + 1,$$

so values of $0 < \lambda < \infty$ imply

$$p + 1 < \text{degrees of freedom} < p + 1 + N.$$

The degrees of freedom can be used to compare competing models as a measure of their complexity.

How do we estimate the remaining variability σ^2? Our estimator from multiple regression was $RSS/(n-(p+1))$, but we need the *residual degrees of freedom* to do this. Consider smoothed estimates $\hat{\mathbf{y}}$ of observations from the model $\mathbf{y} = \mathbf{f} + \boldsymbol{\varepsilon}$, then (setting $\mathbf{W}_\lambda = \mathbf{W}$ for ease)

$$\begin{aligned}
\mathbb{E}(RSS) &= \mathbb{E}\big((\hat{\mathbf{y}}-\mathbf{y})^{\mathrm{T}}(\hat{\mathbf{y}}-\mathbf{y})\big) \\
&= \mathbb{E}\big(\mathbf{y}^{\mathrm{T}}(\mathbf{W}-\mathbf{I})^{\mathrm{T}}(\mathbf{W}-\mathbf{I})\mathbf{y}\big) \\
&= \mathbf{f}^{\mathrm{T}}(\mathbf{W}-\mathbf{I})^{\mathrm{T}}(\mathbf{W}-\mathbf{I})\mathbf{f} + \sigma^2\mathrm{tr}\big((\mathbf{W}-\mathbf{I})^{\mathrm{T}}(\mathbf{W}-\mathbf{I})\big) \\
&= \|(\mathbf{W}-\mathbf{I})\mathbf{f}\|^2 + \sigma^2\big(\mathrm{tr}(\mathbf{W}\mathbf{W}^{\mathrm{T}}) - 2\mathrm{tr}(\mathbf{W}) + n\big).
\end{aligned}$$

[We used the fact $\mathbb{E}(\mathbf{v}^{\mathrm{T}}\mathbf{A}\mathbf{v}) = \mathbb{E}(\mathbf{v})^{\mathrm{T}}\mathbb{E}(\mathbf{v}) + \mathrm{tr}(\mathbf{A}\mathrm{Cov}(\mathbf{v}))$]. The first term is the squared bias, and, assuming it is small then a natural definition for residual degrees of freedom is

$$\mathrm{tr}(\mathbf{W}\mathbf{W}^{\mathrm{T}}) - 2\mathrm{tr}(\mathbf{W}) + n.$$

To choose the smoothing parameter λ, we rely on cross validation. In particular, we seek to minimize

$$RSS_{cv}(\lambda) = \sum_{i=1}^{n}(y_i - \hat{f}_\lambda^{-(i)}(x_i))^2$$

where $\hat{f}_\lambda^{-(i)}(x_i)$ is the fitted value at x_i using all data *except* (x_i, y_i). There is a convenient formula for this RSS,

$$RSS_{cv}(\lambda) = \sum_{i=1}^{n}\left(\frac{y_i - \hat{f}_\lambda(x_i)}{1 - (\mathbf{W}_\lambda)_{ii}}\right)$$

where $\hat{f}_\lambda(x_i)$ is the spline smoothed version using *all* of the data. This corresponds to *leave-one-out* cross-validation, but there are other versions leaving out more than one at a time.

[R example (Splines3.R)]

12.5 Local Regression

Local polynomial fitting follows polynomial regression, but where the residual weights vary with x. For some positive symmetric kernel function K the local polynomial fit at x minimizes

$$\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1(x_i - x) - \cdots - \beta_p(x_i - x)^p)K\left(\frac{x_i - x}{b}\right)$$

where b is the *bandwidth.* The estimated curve is the value $\hat{\beta}_0$ where we use weighted least squares estimates

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^{\mathrm{T}}\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{W}\mathbf{y}$$

where

$$\mathbf{X} = \begin{pmatrix} 1 & x_1 - x & \cdots & (x_1 - x)^p \\ 1 & x_2 - x & \cdots & (x_2 - x)^p \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n - x & \cdots & (x_n - x)^p \end{pmatrix},$$

and

$$\mathbf{W} = \operatorname{diag}\left(K\left(\frac{x_1 - x}{b}\right), \cdots, K\left(\frac{x_n - x}{b}\right)\right).$$

Note that the estimated coefficients *vary with* x, so the locally estimated curve is

$$\hat{f}(x; p, b) = \mathbf{e}_1^{\mathrm{T}}\hat{\boldsymbol{\beta}}.$$

where $\mathbf{e}_1 = (1\,0\,0\cdots 0)^{\mathrm{T}}$. Usually $p = 2$ is sufficiently flexible, and b is estimated by cross-validation or visual inspection. If we set $p = 0$, we get the *Nadaraya-Watson* estimator

$$\hat{f}(x; 0, b) = \frac{\sum_{i=1}^{n} K\left(\frac{x_i - x}{b}\right) y_i}{\sum_{i=1}^{n} K\left(\frac{x_i - x}{b}\right)}.$$

This is a *locally constant* estimator that has been studied since the 60s.

12.6 Generalized Additive Models

A *generalized additive model* (GAM) is the generalization from a regression (classification) problem using a single linear features to many features that may exhibit nonlinear relationships with the response.

Previously, if Y was the response and X the feature, we modeled

$$Y = f(X) + \varepsilon$$

where $f(\cdot)$ was a potentially nonlinear function, such as a spline. Now, for features $X_1, \ldots, X_p$, the GAM model is

$$Y = f_1(X_1) + f_2(X_2) + \cdots f_p(X_p) + \varepsilon$$

where f_i are smooth nonlinear functions, such as splines. Note that this is essentially one way to write the model

$$Y = f(X_1, X_2, \ldots, X_p) + \varepsilon$$

where f can be written as $f_1 + \cdots + f_p$.

Suppose $p = 2$ where we have the two predictors X and Z. If we use a penalized regression spline basis for $f_1(x)$ and $f_2(z)$ of the form

$$f_1(x) = \beta_0 + \beta_1 x + \sum_{i=1}^{p_x} \beta_{1i} k(x, b_i)$$

and

$$f_2(z) = \alpha_0 + \alpha_1 z + \sum_{i=1}^{p_z} \alpha_{1i} k(z, a_i)$$

where $f_1(x)$ has p_x knots $b_1, \ldots, b_{p_x}$ and $f_2(z)$ has p_z knots $a_1, \ldots, a_{p_z}$, then there is clearly an identifiability problem in

$$Y = f_1(X) + f_2(Z) + \varepsilon$$

with β_0 and α_0. Setting $\alpha_0 = 0$ relieves this, and then we can write

$$\int_0^1 f_1''(x)^2 \mathrm{d}x = \boldsymbol{\beta}^{\mathrm{T}} \mathbf{S}_x \boldsymbol{\beta}$$

and

$$\int_0^1 f_2''(z)^2 \mathrm{d}z = \boldsymbol{\alpha}^{\mathrm{T}} \mathbf{S}_z \boldsymbol{\alpha}$$

for $\mathbf{S}_x$ and $\mathbf{S}_z$ having the first two rows and columns zero, and the $(i+2, j+2)$th slot $k(b_i, b_j)$ and $k(a_i, a_j)$, respectively.

With this setup, the penalized model is the minimizing solution to

$$\sum_{i=1}^{n}(y_i-(f_1(x_i)+f_2(z_i)))^2+\lambda_1\int_0^1 f_1''(x)^2\mathrm{d}x+\lambda_2\int_0^1 f_2''(z)^2\mathrm{d}z$$

which can equivalently be written

$$(\mathbf{y}-\mathbf{X}\boldsymbol{\theta})^{\mathrm{T}}(\mathbf{y}-\mathbf{X}\boldsymbol{\theta})+\boldsymbol{\theta}^{\mathrm{T}}\mathbf{S}\boldsymbol{\theta}$$

where $\boldsymbol{\theta}=(\beta_0,\ldots,\beta_{p_x},\alpha_1,\ldots,\alpha_{p_z})^{\mathrm{T}}$ and $\mathbf{X}$ is the design matrix involving $1, x_i, z_i, k(z_i,b_j), k(x_i,b_j)$ and

$$\mathbf{S}=\begin{pmatrix}\mathbf{S}_x & \mathbf{0}\\ \mathbf{0} & \mathbf{S}_z\end{pmatrix}.$$

Fitting a GAM for the non-Gaussian case is nontrivial, and uses a method known as *back-fitting*, repeatedly updating the fit for each predictor holding all others fixed. GAMs allow nonlinear relationships and convenient interpretations (e.g., we can examine the smoothness for each fitted f_j by its degrees of freedom), but they are additive and miss out on interactions.

[R example: (GAMs.R)]

12.7 Smoothing Splines Revisited

Some argue that penalized splines are more flexible than smoothing splines due to the ability to select knots as well as a sensible penalty. Indeed, sometimes penalized splines will be referred to as *low rank* since they reduce the number of knots below n, whereas smoothing splines are *full rank* since they place a knot at every x_i coordinate. However, there is another interpretation of the smoothing spline solution as *lying in a particular class of functions.*

Recall the cubic smoothing spline problem, if $\mathcal{H}=\{f|\int_0^1 f''(x)^2\mathrm{d}x<\infty\}$, then the cubic smoothing spline solution is the minimizer of

$$\sum_{i=1}^{n}(y_i-f(x_i))^2+\lambda\int_0^1 f''(x)^2\mathrm{d}x,$$

over $f\in\mathcal{H}$. We will show later that this solution can be written

$$\hat{f}(x)=\alpha_0+\alpha_1 x+\sum_{i=1}^{n}\beta_i k(x,x_i)$$

where

$$k(u,v) = \begin{cases} \frac{1}{2}u^2v - \frac{1}{6}u^3, & u < v \\ \frac{1}{2}v^2u - \frac{1}{6}v^3, & u \geq v \end{cases}$$

and, moreover, $\mathcal{H}$ is the space of functions that is the completion of all linear combination of k, $\sum_{i=1}^{\infty} a_i k(x, z_i)$.

13 Gaussian Process Regression

This section relies heavily on the multivariate normal distribution. Recall the definition:

Definition 26. *We say the random vector* $\mathbf{X} = (X_1, X_2, \ldots, X_n)^{\mathrm{T}}$ *has a* multivariate normal *distribution if*

$$f(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^n \det \Sigma}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)$$

for $\mathbf{x} \in \mathbb{R}^n$*. We often write* $\mathbf{X} \sim MVN(\boldsymbol{\mu}, \Sigma), \mathbf{X} \sim N_n(\boldsymbol{\mu}, \Sigma)$ *or* $\mathbf{X} \sim N(\boldsymbol{\mu}, \Sigma)$*.*

Here, $\mathbb{E}\mathbf{X} = \boldsymbol{\mu}$ is the vector of means (i.e., $\mathbb{E}X_i = \mu_i$), and $\mathrm{Cov}(\mathbf{X}, \mathbf{X}) = \Sigma$ is the covariance matrix (sometimes called variance-covariance matrix). The (i, j)th entry of Σ is $\mathrm{Cov}(X_i, X_j)$.

[Imagine this as two parameters, $\boldsymbol{\mu}$ tell us the average behavior of the vector, and Σ tells us how all elements relate to each other]

[Picture of bivariate density with projected contours]

The covariance matrix is *symmetric* since $\Sigma_{ij} = \mathrm{Cov}(X_i, X_j) = \mathrm{Cov}(X_j, X_i) = \Sigma_{ji}$. The diagonal of Σ contains the variances of each component. Additionally, we have

- Each X_i is marginally normally distributed: $X_i \sim N(\mu_i, \sigma_i^2)$
- If A is a $k \times n$ real matrix and $\mathbf{b} \in \mathbb{R}^k$, then $A\mathbf{X} + \mathbf{b} \sim MVN(A\boldsymbol{\mu} + \mathbf{b}, A\Sigma A^{\mathrm{T}})$
- If Σ is a covariance matrix then it is nonnegative definite
- If Σ is nonnegative definite, then it is a covariance matrix.

[R example (MultivariateNormal.R)]

Our basic model is

$$Y = f(X) + \varepsilon$$

where, until now, we have assumed some parametric or nonparametric form for f based on X. In this section we will assume f is a *random function* from some particular *class of functions.*

Definition 27. *A* Gaussian process *$f(x)$ on $x \in \mathbb{R}$ is a random function such that $(f(x_1), \ldots, f(x_n))^{\mathrm{T}}$ is multivariate normal for any choices of $x_1, \ldots, x_n$.*

Example 28. *If $f(x)$ is $N(0,1)$ and $f(x)$ and $f(y)$ are independent for any $x \neq y$, then $f(x)$ is a Gaussian process. For example, let $x_1, x_2, x_3 \in \mathbb{R}$, then*

$$\begin{pmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \end{pmatrix} \sim N\left(\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \right).$$

[Picture]

[Clearly this isn't a very good candidate for a nice looking "random function"]

13.1 Building the GP Framework

Thinking of f as a random function is equivalent to putting a prior distribution on f. To motivate a general Gaussian process, suppose we have features $X_1, \ldots, X_p$ and statistical model

$$Y = f(\mathbf{X}) + \varepsilon = \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon$$

where ε are iid $N(0, \sigma^2)$ random variables (we can consider $X_1 = 1$ for an intercept, or remove it using $\bar{y}$). Consider a prior distribution on $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)^{\mathrm{T}} \sim N(\mathbf{0}, \Sigma)$ where

$$\Sigma_{ij} = \mathrm{Cov}(\beta_i, \beta_j), \quad i, j = 1, \ldots, p$$

is the prior covariance matrix. This puts a prior distribution on f as

$$f(\mathbf{x}) \sim N(0, \mathbf{x}^{\mathrm{T}} \Sigma \mathbf{x}),$$

or

$$\mathbf{f} \sim N(\mathbf{0}, \mathbf{X} \Sigma \mathbf{X}^{\mathrm{T}})$$

where $\mathbf{f} = (f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n))^{\mathrm{T}}$ and $\mathbf{X} = [\mathbf{x}_1\, \mathbf{x}_2 \cdots \mathbf{x}_n]^{\mathrm{T}}$. It's important to note here that $\mathbf{X}$ is only rank p, and so is $\mathbf{X}\Sigma\mathbf{X}^{\mathrm{T}}$, which puts constraints on the relationships between the components of $\mathbf{f}$.

Given data $(y_1, \mathbf{x}_1), \ldots, (y_n, \mathbf{x}_n)$ the *observational model* is $[\mathbf{y}|\boldsymbol{\beta}] \sim N(\mathbf{X}\boldsymbol{\beta}, \sigma^2\mathbf{I})$. The typical goal of a Bayesian analysis is to summarize our uncertainty in $\boldsymbol{\beta}$ given the data $\mathbf{y}$, i.e., derive the *posterior distribution* $[\boldsymbol{\beta}|\mathbf{y}]$ as the *posterior density*

$$p(\boldsymbol{\beta}|\mathbf{y}) = \frac{p(\mathbf{y}|\boldsymbol{\beta})\pi(\boldsymbol{\beta})}{p(\mathbf{y})}$$

where π is the *prior density* for $\boldsymbol{\beta}$ and $p(\mathbf{y})$ is the marginal likelihood integrating out uncertainty in $\boldsymbol{\beta}$:

$$p(\mathbf{y}) = \int p(\mathbf{y}, \boldsymbol{\beta})\mathrm{d}\boldsymbol{\beta} = \int p(\mathbf{y}|\boldsymbol{\beta})\pi(\boldsymbol{\beta})\mathrm{d}\boldsymbol{\beta}.$$

It's not too hard to show that under this setup the posterior distribution for $\boldsymbol{\beta}$ is normal,

$$[\boldsymbol{\beta}|\mathbf{y}] \sim N\left(\sigma^{-2}(\sigma^{-2}\mathbf{X}^{\mathrm{T}}\mathbf{X} + \Sigma^{-1})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}, (\sigma^{-2}\mathbf{X}^{\mathrm{T}}\mathbf{X} + \Sigma^{-1})^{-1}\right) = N(\boldsymbol{\beta}_{post}, \Sigma_{post})$$

[Thought experiment: what happens when the prior *precision* Σ^{-1} is small; what happens when the measurement variance σ^2 is large?] Note we can write

$$\boldsymbol{\beta}_{post} = (\mathbf{X}^{\mathrm{T}}\mathbf{X} + \sigma^2\Sigma^{-1})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

so setting $\Sigma = \mathbf{I}$ we have that the posterior mode is the same as the ridge regression estimator with $\lambda = \sigma^2$.

For prediction at a new feature setting $\mathbf{x}_*$ the *posterior predictive distribution* is the distribution $p(f(\mathbf{x}_*)|\mathbf{y}) = \int p(f(\mathbf{x}_*)|\boldsymbol{\beta}, \mathbf{y})p(\boldsymbol{\beta}|\mathbf{y})\mathrm{d}\boldsymbol{\beta}$. For our setup, we have

$$[f(\mathbf{x}_*)|\mathbf{y}] \sim N(\mathbf{x}_*^{\mathrm{T}}\boldsymbol{\beta}_{post}, \mathbf{x}_*^{\mathrm{T}}\Sigma_{post}\mathbf{x}_*)$$

or in general

$$[\mathbf{f}|\mathbf{y}] \sim N\left(\mathbf{X}\boldsymbol{\beta}_{post}, \mathbf{X}\Sigma_{post}\mathbf{X}^{\mathrm{T}}\right).$$

Again we should recognize that if $\mathbf{X}$ is of rank $p < n$ then $\mathbf{X}\Sigma_{post}\mathbf{X}^{\mathrm{T}}$ has $n-p$ zero eigenvalues, putting restrictions on possible outcomes of $\mathbf{f}$.

Of course we know the multiple regression model for $f(\mathbf{x})$ is often insufficiently flexible, so we might consider

$$Y = \beta_1 f_1(X_1) + \beta_2 f_2(X_1) + \cdots + \beta_N f_N(X_p) + \varepsilon$$

for some transformation functions (e.g. splines) of a set of original features. Set the notation

$$\boldsymbol{\phi}(\mathbf{x}) = (f_1(x_1), f_2(x_1), \ldots, f_N(x_p))^{\mathrm{T}}$$

that is the mapping $\boldsymbol{\phi} : \mathbb{R}^p \to \mathbb{R}^N$ into a larger space of covariates. Then our implied model is

$$f(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\beta}$$

where now $\boldsymbol{\beta}$ is a $N \times 1$ vector. The observational model is

$$\mathbf{Y} = \boldsymbol{\Phi}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

where $\boldsymbol{\Phi} = [\boldsymbol{\phi}(\mathbf{x}_1)\, \boldsymbol{\phi}(\mathbf{x}_2) \cdots \boldsymbol{\phi}(\mathbf{x}_n)]^{\mathrm{T}}$ is a $n \times N$ matrix of transformed features. Then with the previous prior, $\boldsymbol{\beta} \sim N(\mathbf{0}, \Sigma)$ we have

$$[\boldsymbol{\beta}|\mathbf{y}] \sim N\left(\sigma^{-2}(\sigma^{-2}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \Sigma^{-1})^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{y}, (\sigma^{-2}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \Sigma^{-1})^{-1}\right)$$

(which is the same as previously replacing $\mathbf{X}$ by $\boldsymbol{\Phi}$). Finally, the predictive distribution at new feature vector $\mathbf{x}_*$ is

$$[f(\mathbf{x}_*)|\mathbf{y}] \sim N\left(\sigma^{-2}\boldsymbol{\phi}(\mathbf{x}_*)^{\mathrm{T}}(\sigma^{-2}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \Sigma^{-1})^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{y}, \boldsymbol{\phi}(\mathbf{x}_*)^{\mathrm{T}}(\sigma^{-2}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \Sigma^{-1})^{-1}\boldsymbol{\phi}(\mathbf{x}_*)\right).$$

[R example (GPIntro.R)]

13.2 A General Gaussian Process Regression Framework

We'd like to allow for "more general" random functions than those specified by a few transformations $\boldsymbol{\phi}(x)$, but how should these be parameterized? [Thought experiment: $f(x) \sim N(0, 1)$, consider how to specify the distribution of $(f(0), f(0.1), f(5))^{\mathrm{T}}$ in such a way that enforces some sort of regularity]

For exposition, suppose f is a Gaussian process on $\mathbb{R}$. A Gaussian process (GP) is specified by its

- *Mean function* $\mu(x) = \mathbb{E}f(x)$ and
- *Covariance function* or *covariance kernel* $k(x, y) = \mathrm{Cov}(f(x), f(y))$.

These specify the full model since

$$\begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{pmatrix} \sim N\left(\begin{pmatrix} \mu(x_1) \\ \mu(x_2) \\ \vdots \\ \mu(x_n) \end{pmatrix}, \begin{pmatrix} k(x_1,x_1) & k(x_1,x_2) & \cdots & k(x_1,x_n) \\ k(x_2,x_1) & k(x_2,x_2) & \cdots & k(x_2,x_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_n,x_1) & k(x_n,x_2) & \cdots & k(x_n,x_n) \end{pmatrix} \right)$$

for any choices of $x_1, \ldots, x_n$. The covariance matrix here is called a *Gram matrix*. [A general GP is specified by μ and k, but this is still too general for use in applications]

[Picture of GPs with different means/covariances]

Gaussian process regression

Suppose f is a Gaussian process on $\mathbb{R}$ with mean zero and covariance kernel $k(\cdot,\cdot)$. Moreover, suppose we have observations $\mathbf{y} = (y_1, \ldots, y_n)^{\mathrm{T}}$ with corresponding features $\mathbf{x} = (x_1, \ldots, x_n)^{\mathrm{T}}$.

Suppose interest focuses on f at a particular set of features, $\mathbf{f}_* = (f(x_{1*}), f(x_{2*}), \ldots, f(x_{m*}))^{\mathrm{T}}$. Then for GPs we have

$$\begin{pmatrix} \mathbf{f}_* \\ \mathbf{y} \end{pmatrix} \sim N\left(\begin{pmatrix} \mathbf{0} \\ \mathbf{0} \end{pmatrix}, \begin{pmatrix} \mathrm{Cov}(\mathbf{f}_*, \mathbf{f}_*) & \mathrm{Cov}(\mathbf{f}_*, \mathbf{y}) \\ \mathrm{Cov}(\mathbf{y}, \mathbf{f}_*) & \mathrm{Cov}(\mathbf{y}, \mathbf{y}) \end{pmatrix} \right).$$

What does GP regression look like when we want $[\mathbf{f}_*|\mathbf{y}]$? We know for multivariate normals that

$$[\mathbf{f}_*|\mathbf{y}] \sim N\left(\mathrm{Cov}(\mathbf{f}_*, \mathbf{y})\mathrm{Cov}(\mathbf{y}, \mathbf{y})^{-1}\mathbf{y}, \mathrm{Cov}(\mathbf{f}_*, \mathbf{f}_*) - \mathrm{Cov}(\mathbf{f}_*, \mathbf{y})\mathrm{Cov}(\mathbf{y}, \mathbf{y})^{-1}\mathrm{Cov}(\mathbf{y}, \mathbf{f}_*)\right).$$

Let's specialize the above to the case $x_{i*} = x_i$, that is, finding the fitted values. Do so first for the model

$$Y = f(X),$$

that is, GP regression without error (where we perfectly observed the random function at features $x_1, \ldots, x_n$). Then $\mathbf{f}_* = \mathbf{y}$ and we have

$$[\mathbf{f}_*|\mathbf{y}] \sim N(\mathbf{y}, \mathbf{0}),$$

that is, the fitted values are just the observations. Contrast with the usual model

$$Y = f(X) + \varepsilon$$

where we observe a noisy version of the regression where we now suppose ε are iid $N(0, \tau^2)$ errors. Then because $\mathbf{y} = \mathbf{f} + \boldsymbol{\varepsilon}$ we have

$$\mathrm{Cov}(\mathbf{y}, \mathbf{y}) = \mathrm{Cov}(\mathbf{f}, \mathbf{f}) + \mathrm{Cov}(\boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}) = \Sigma + \tau^2\mathbf{I},$$

where $\Sigma = (k(x_i, x_j))_{i,j=1}^n$ is the covariance matrix parameterized by k. Thus,

$$[\mathbf{f}_*|\mathbf{y}] \sim N\left(\Sigma(\Sigma + \tau^2\mathbf{I})^{-1}\mathbf{y}, \Sigma - \Sigma(\Sigma + \tau^2\mathbf{I})^{-1}\Sigma\right).$$

Note that the conditional mean depends on $\mathbf{y}$, but the conditional variance depends only on Σ and τ^2. In particular, $\Sigma = (k(x_i, x_j))_{i,j=1}^n$ depends on the choice of covariance function, and thus has significant control over the conditional mean and variance.

Parameterizing mean and covariance functions

We've already seen some GP specifications: if $f(x) = \boldsymbol{\phi}(x)^{\mathrm{T}}\boldsymbol{\beta}$ with $\boldsymbol{\beta} \sim N(\mathbf{0}, \Sigma)$ then

$$\begin{aligned} \mu(x) &= 0 \\ k(x, z) &= \boldsymbol{\phi}(x)^{\mathrm{T}}\Sigma\boldsymbol{\phi}(z). \end{aligned}$$

Mean functions are typically specified as regressions; for now we will assume $\mu(x) = 0$.

A simple example of a covariance comes from the case where

$$f(X) = \beta_0 + \beta_1 X$$

with a $N(0, \sigma_1^2)$ prior on β_0 and a $N(0, 1)$ prior on β_1; the implied covariance is

$$k(x, z) = \sigma_1^2 + xz.$$

Another example is the polynomial kernel,

$$k(x, z) = (\sigma_1^2 + xz)^p$$

for some $p > 0$. These covariances are invariant to a rotation about the origin, but not translations in feature space.

Covariance functions are often assumed to be *stationary* in that $k(x, z) = k(x - z) = k(r)$. We seek candidate classes of covariances that depend only on a small number of parameters. Some of the most common are:

- Exponential:
$$K(r) = \sigma^2 \exp\left(-\frac{r}{a}\right), \qquad a > 0$$
- Squared exponential or Gaussian:
$$K(r) = \sigma^2 \exp\left(-\frac{r^2}{a}\right), \qquad a > 0$$
- Matérn:
$$K(r) = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{r}{a}\right)^{\nu} K_\nu\left(\frac{r}{a}\right), \qquad a > 0, \nu > 0,$$
where K_ν is a modified Bessel function of the second kind of order ν.

In the above, σ^2 is a variance parameter, a is called a range (or sometimes scale) parameter that determines the rate of decay of the covariance function. The ν in the Matérn covariance is a smoothness parameter. The Matérn contains the exponential and Gaussian as special cases, specifically when $\nu = 0.5$ and $\nu \to \infty$, respectively.

[Picture of candidate covariances]

An important notion for covariance functions is *mean square differentiability.* In particular, we say $f(x)$ is mean square continuous at x if
$$\lim_{z \to x} \mathbb{E}(f(x) - f(z))^2 = 0$$
and is MS continuous if it is MS continuous at every point. It is said to be MS differentiable at x if
$$\lim_{h \to 0} \mathbb{E}\left(\frac{f(x+h) - f(x)}{h}\right)^2$$
exists. Note that this is equivalent to saying that k has a derivative at 0. For instance, squared exponential covariances imply infinitely many MS derivatives, while Matérn covariances imply d derivatives only when $\nu > d$.

If there is a vector of features, $\mathbf{x} = (x_1, \ldots, x_p)^{\mathrm{T}}$ then the easiest way to form valid covariances is using *separable kernels*, e.g.,
$$k(\mathbf{x}, \mathbf{z}) = k_1(x_1, z_1) \cdots k_p(x_p, z_p),$$

where $k_1, \ldots, k_p$ are valid covariances. For example,

$$k(\mathbf{x}, \mathbf{z}) = \exp\left(\left(\frac{x_1 - z_1}{a_1}\right)^2\right) \cdots \exp\left(\left(\frac{x_p - z_p}{a_p}\right)^2\right)$$

is a separable squared exponential kernel.

[R example (GPSimulations.R)]

Linking the finite and infinite dimensional GP regressions

Note the way the covariates enter into the posterior distributions: products like

$$\boldsymbol{\phi}(\mathbf{x}_1)^{\mathrm{T}} \Sigma \boldsymbol{\phi}(\mathbf{x}_2) = \sum_{i=1}^{N} \sum_{j=1}^{N} \phi_i(\mathbf{x}_1) \phi_j(\mathbf{x}_2) \Sigma_{ij}.$$

Since Σ is positive definite it has SVD $\Sigma = \mathbf{U}\mathbf{D}\mathbf{U}^{\mathrm{T}}$, so

$$\boldsymbol{\phi}(\mathbf{x}_1)^{\mathrm{T}} \Sigma \boldsymbol{\phi}(\mathbf{x}_2) = \boldsymbol{\psi}(\mathbf{x}_1)^{\mathrm{T}} \mathbf{D} \boldsymbol{\psi}(\mathbf{x}_2) = \sum_{i=1}^{N} \lambda_i \psi_i(\mathbf{x}_1) \psi_i(\mathbf{x}_2)$$

for $\boldsymbol{\psi}(\mathbf{x}) = \mathbf{U}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x})$ and $\mathbf{D} = \mathrm{diag}(\lambda_1, \ldots, \lambda_N)$ with $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$. As $N \to \infty$ we will write

$$\sum_{i=1}^{\infty} \lambda_i \psi_i(\mathbf{x}_1) \psi_i(\mathbf{x}_2) = k(\mathbf{x}_1, \mathbf{x}_2) \tag{10}$$

where $k : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$ is a positive definite function. In fact, Mercer's theorem states the converse: *any* positive definite function can be decomposed as (10). For example, the squared exponential $k(x, z) = \exp(-(x - z)^2)$ is the limiting covariance for an infinite regression with iid normal priors on squared exponential kernels centered at a dense set of features over the reals.

[R example (GPRegression.R)]

There are (at least) two viewpoints of what a Gaussian process regression function is.

Viewpoint 1: It is the posterior mode under a Gaussian observational model with Gaussian prior on the coefficients of some finite- or infinite-dimensional regression model.

Viewpoint 2: It is a penalized least squares regression solution where the penalty is acting on a *space of functions*. This space of functions is usually a *reproducing kernel Hilbert space* where the reproducing kernel is exactly the covariance function from the alternative viewpoint.

Why is it useful to discuss both viewpoints? It all comes down to how you want to interpret your model and the ultimate goals. The Bayesian framework is useful if you want to produce confidence regions and predictive standard errors; however it is not always clear to which class of functions these priors correspond (e.g., what does an exponential covariance prior mean?). On the other hand, if you wanted to use a cubic spline-like penalty that explicitly penalizes wiggliness, which corresponding covariance model are you using from a Bayesian point of view (it turns out you would never invent it in a vacuum), and how might you go about creating predictive confidence intervals?

13.3 Reproducing kernel Hilbert spaces

We need a few elementary definitions to proceed.

Definition 29. *We say* $\langle \cdot, \cdot \rangle : \mathcal{H} \times \mathcal{H} \to \mathbb{R}$ *is an* inner product *over the vector space* $\mathcal{H}$ *if*

- $\langle f, g \rangle = \langle g, f \rangle$
- $\langle af, g \rangle = a\langle f, g \rangle$ *for any* $a \in \mathbb{R}$ *and* $\langle f + h, g \rangle = \langle f, g \rangle + \langle h, g \rangle$ *and*
- $\langle f, f \rangle \geq 0$ *and if* $\langle f, f \rangle = 0$ *then* $f \equiv 0$.

Definition 30. *A vector space* $\mathcal{H}$ *is a* Hilbert space *if it is a complete metric space with respect to the distance*

$$\|f - g\| = \sqrt{\langle f - g, f - g \rangle}.$$

Recall that completeness means that $\mathcal{H}$ contains all of its limit points. The connection to GP regression uses Hilbert spaces of functions, that is, spaces where each element is a function.

Definition 31. *A function* $k : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ *is a* reproducing kernel *of the Hilbert space* $\mathcal{H}$ *if*

- *$k(\cdot, x) \in \mathcal{H}$ for all x*
- *$\langle f, k(\cdot, x)\rangle = f(x)$ for all $f \in \mathcal{H}$ and all x.*

Note that $k(\cdot, x)$ *reproduces* f at a location x. Note that

$$\langle k(\cdot, x), k(\cdot, y)\rangle = k(x, y)$$

and

$$k(x, y) = \langle k(\cdot, x), k(\cdot, y)\rangle = \langle k(\cdot, y), k(\cdot, x)\rangle = k(y, x)$$

so that k is a symmetric function. A Hilbert space admitting a reproducing kernel is known as a *reproducing kernel Hilbert space* (RKHS).

Example 32. *Suppose $\{e_1, \ldots, e_n\}$ is an orthonormal basis for a Hilbert space $\mathcal{H}$. Then*

$$k(x, y) = \sum_{i=1}^{n} e_i(x)e_i(y)$$

is a reproducing kernel for $\mathcal{H}$. To check, first note

$$k(x, \cdot) = \sum_{i=1}^{n} e_i(x)e_i(\cdot) = \sum_{i=1}^{n} \lambda_i e_i(\cdot)$$

is an element of $\mathcal{H}$ by the definition of a basis. Additionally, any $\phi \in \mathcal{H}$ can be written $\phi(x) = \sum_{i=1}^{n} c_i e_i(x)$, so

$$\begin{aligned}\langle \phi(\cdot), k(\cdot, x)\rangle &= \left\langle \sum_{i=1}^{n} c_i e_i(\cdot), k(\cdot, x) \right\rangle \\ &= \sum_{i=1}^{n} c_i \langle e_i(\cdot), k(\cdot, x)\rangle \\ &= \sum_{i=1}^{n} c_i e_i(x) = \phi(x).\end{aligned}$$

Thus, any finite-dimensional Hilbert space is a RKHS.

The following two theorems identify a reproducing kernel with a positive definite function, that is, they are one in the same.

Theorem 33. *If $\mathcal{H}$ is a RKHS with reproducing kernel $k(\cdot,\cdot)$, then k is a positive definite function.*

Proof. For any $a_1,\ldots,a_n \in \mathbb{R}$ and any $x_1,\ldots,x_n \in \mathbb{R}$, we have

$$\sum_{i=1}^n \sum_{j=1}^n a_i a_j k(x_i,x_j) = \left(\sum_{i=1}^n a_i k(\cdot,x_i)\right)^2 \geq 0.$$

$\square$

Theorem 34 (Moore-Aronszajn). *If $k(\cdot,\cdot)$ is a positive definite function on $\mathbb{R}\times\mathbb{R}$ then there exists a* unique *RKHS $\mathcal{H}$ having k as its reproducing kernel. Moreover, the functions $f(x)=\sum_{i=1}^n a_i k(x,x_i)$ are dense in $\mathcal{H}$.*

13.4 The Gaussian process smoother

Given a set of data $(x_1,y_1),\ldots,(x_n,y_n)$ the *Gaussian process regression function* $\hat{f}(x)$ is the function that minimizes

$$\begin{aligned}\mathcal{L}(f) &= \sum_{i=1}^n (y(x_i)-f(x_i))^2 + \lambda\langle f,f\rangle \\ &= \sum_{i=1}^n (y(x_i)-f(x_i))^2 + \lambda\|f\|^2\end{aligned}$$

over the class of functions $\mathcal{H} = \{f \mid \|f\| < \infty\}$.

It turns out the minimizing solution can be written

$$\hat{f}(x) = \sum_{i=1}^n m_i k(x,x_i)$$

where $\mathbf{m}$ is the minimizing solution of

$$\|\mathbf{y}-\Sigma\mathbf{m}\|^2 + \lambda\mathbf{m}^{\mathrm{T}}\Sigma\mathbf{m}$$

with $\mathbf{m}=(m_1,\ldots,m_n)^{\mathrm{T}}$ and $\Sigma = (k(x_i,x_j))_{i,j=1}^n$. In particular,

$$\mathbf{m} = (\Sigma+\lambda I)^{-1}\mathbf{y}.$$

Thus,

$$\begin{aligned}\hat{\mathbf{f}} &= (\hat{f}(x_1), \ldots, \hat{f}(x_n))^{\mathrm{T}} \\ &= \Sigma(\Sigma + \lambda I)^{-1}\mathbf{y}\end{aligned}$$

and if $\lambda = 0$ we interpolate the data, while if $\lambda \to \infty$, $\hat{f}(x) \to 0$.