Brian Steele

M 462/562 Notes

January 14, 2019

Springer

Contents

Chapter 1
Introduction

The broad objective of M462/562 is a study of predictive analytics. Predictive analytics encompasses the problem of predicting an unobserved realization of a process. The subject also encompasses a number of analytical methods aimed at solving problems encountered in the prediction problem, however. More broadly, predictive analytics includes a variety of methods used for and supporting the data analysis for prediction. Many of these methods originate from statistics and computer science though in those fields the subject matter is commonly referred to as statistical learning and machine learning, respectively.

There are three major sub-areas within the subject area of predictive analytics: dimension reduction, unsupervised classification, and supervised classification. One might even argue that artificial neural networks deserves to stand alone as a fourth sub-area given its prominence, technological importance, and the wide philosophical gap between predictive analytics and statistical learning. In this course, all four topics are explored from a mathematical perspective and with the aim of understanding the foundations (which are mathematical). For most students, programming an algorithm is the surest way of understanding it (after having processed the mathematics) and so there is more emphasis on computation and programming herein than most mathematical texts.

We begin with establishing mathematical notation.

1.1 Notation

A vector consisting of p elements is denoted as

$$\underset{p\times 1}{\mathbf{y}} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{bmatrix}.$$

A vector may be thought of as a $p \times 1$ matrix. The transpose of $\mathbf{y}$ is a row-vector or, equivalently, $1 \times p$ matrix. The row vector yf is written as $\mathbf{y} = [y_1 \quad y_2 \quad \cdots \quad y_p]^T$. A matrix is two-dimensional array of scalars, and so the elements are real numbers.[1] To show the individual elements comprising the matrix, we write

$$\underset{n\times p}{\mathbf{Y}} = \begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,p} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n,1} & y_{n,2} & \cdots & y_{n,p} \end{bmatrix}. \tag{1.1}$$

Our convention for subscripting sets uses the left subscript to identify the row position and the right subscript to identify the column position of the scalar $y_{i,j}$. Thus, $y_{i,j}$ occupies row i and column j. If the matrix is neither a column nor a row vector, then the symbol representing the matrix is written in upper case and in bold. We also write $\mathbf{Y} = [y_{i,j}]$ as a means of showing the notation for both the matrix and its elements.

A set of row vectors is sometimes stacked to form a matrix. For example, $\mathbf{Y}$ (formula 1.1) may be expressed as

$$\underset{n\times p}{\mathbf{Y}} = \begin{bmatrix} \mathbf{y}_1^T \\ \mathbf{y}_2^T \\ \vdots \\ \mathbf{y}_n^T \end{bmatrix},$$

where $\mathbf{y}_i^T$ is the ith row of $\mathbf{Y}$. Likewise, a set of column vectors may be concatenated to form a matrix. It's necessary at times to explicitly denote to the rows of a matrix and the columns of a matrix, and so we introduce the following notations:

$$\underset{n\times p}{\mathbf{Y}} = \begin{bmatrix} \mathbf{y}_{\cdot 1} \\ \mathbf{y}_{\cdot 2} \\ \vdots \\ \mathbf{y}_{\cdot n} \end{bmatrix} = \begin{bmatrix} \mathbf{y}_{1\cdot} & \mathbf{y}_{2\cdot} & \cdots & \mathbf{y}_{n\cdot} \end{bmatrix}.$$

Thus, $\mathbf{y}_{\cdot i}$ is a $1 \times p$ *row* vector—specifically, row i of $\mathbf{Y}$, and $\mathbf{y}_{\cdot j}$ is a $n \times 1$ *column* vector—specifically, column j of $\mathbf{Y}$. The position of the dot in the

[1] Sometimes the elements are constrained to be integers.

subscript provides a reminder of whether the object is a row or a column vector.

The product of a scalar $a \in \mathbb{R}$ and a matrix $\mathbf{B}$ is computed by multiplying every element in the matrix $\mathbf{B}$ by a. For example, $a\underset{1\times q}{\mathbf{b}^T} = [ab_1 \;\; ab_2 \;\; \cdots \;\; ab_q]$. Two frequently used vector products are the inner product and the outer product. The inner product of $\mathbf{x}$ and $\mathbf{y}$ is

$$\underset{1\times p}{\mathbf{x}^T}\underset{p\times 1}{\mathbf{y}} = \sum_{i=1}^{p} x_i y_i = \mathbf{y}^T\mathbf{x}.$$

The inner product is defined only if the lengths of the two vectors are equal, in which case, the vectors are said to be *conformable*.

The product of two conformable matrices $\mathbf{A}$ and $\mathbf{B}$ is

$$\underset{p\times n}{\mathbf{A}}\underset{n\times q}{\mathbf{B}} = \underset{p\times q}{\begin{bmatrix} \mathbf{a}_{1\cdot}^T\mathbf{b}_{\cdot 1} & \cdots & \mathbf{a}_{1\cdot}^T\mathbf{b}_{\cdot q} \\ \mathbf{a}_{2\cdot}^T\mathbf{b}_{\cdot 1} & \cdots & \mathbf{a}_{2\cdot}^T\mathbf{b}_{\cdot q} \\ \vdots & \ddots & \vdots \\ \mathbf{a}_{p\cdot}^T\mathbf{b}_{\cdot 1} & \cdots & \mathbf{a}_{p\cdot}^T\mathbf{b}_{\cdot q} \end{bmatrix}}, \tag{1.2}$$

where $\mathbf{a}_{i\cdot}^T$ is the ith row of $\mathbf{A}$ and $\mathbf{b}_{\cdot j}$ is the jth column of $\mathbf{B}$.

The outer product of a vector $\mathbf{x}$ with another vector $\mathbf{w}$ is a matrix given by

$$\underset{p\times 1}{\mathbf{x}}\underset{1\times q}{\mathbf{w}^T} = \begin{bmatrix} x_1w_1 & \cdots & x_1w_q \\ x_2w_1 & \cdots & x_2w_q \\ \vdots & \ddots & \vdots \\ x_pw_1 & \cdots & x_pw_q \end{bmatrix}. \tag{1.3}$$

Generally, $\mathbf{x}\mathbf{w}^T \neq \mathbf{w}\mathbf{x}^T$.

If a matrix $\mathbf{A}$ is square and full rank, meaning that the columns of $\mathbf{A}$ are linearly independent, then there exists an *inverse* $\mathbf{A}^{-1}$ of $\mathbf{A}$. The product of $\mathbf{A}$ and its inverse is the identity matrix

$$\underset{n\times n}{\mathbf{I}} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}. \tag{1.4}$$

Hence, $\mathbf{I} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A}$ since the inverse of $\mathbf{A}^{-1}$ is $\mathbf{A}$. If a solution $\mathbf{x}$ is needed to solve the equation

$$\underset{p\times p}{\mathbf{A}}\underset{p\times 1}{\mathbf{x}} = \underset{p\times 1}{\mathbf{y}}, \tag{1.5}$$

and $\mathbf{A}$ is invertible (that is, $\mathbf{A}$ has an inverse), then the solution of the equation is $\mathbf{x} = \mathbf{A}^{-1}\mathbf{y}$.

1.2 Exercises

Grad students should do all problems. Undergraduates should do at least 3.

1. Let
$$\underset{n\times 1}{\mathbf{j}} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$
denote the *summation* vector of length n.
   (a) Show that for any n-vector $\mathbf{x}$, $n^{-1}\mathbf{j}^T\mathbf{x} = \overline{x}$ (the sample of the observations composing $\mathbf{x}$).
   (b) Suppose that the p-vector of sample means $\overline{\mathbf{x}}$ is to be computed from the data matrix $\underset{n\times p}{\mathbf{X}}$. Show that
$$\overline{\mathbf{x}}^T = (\mathbf{j}^T\mathbf{j})^{-1}\,\underset{1\times n}{\mathbf{j}^T}\underset{n\times p}{\mathbf{X}}. \tag{1.6}$$
   (c) Show that $\overline{\mathbf{x}}$ minimizes $(\mathbf{X} - \mathbf{j}\mu^T)^T(\mathbf{X} - \mathbf{j}\mu^T)$ with respect to μ.
2. Given the p-vector $\mathbf{x}$, determine the p-vector $\mathbf{y}$ such that $1 = \mathbf{x}^T\mathbf{y}$.
3. Show that $\overline{y} = \sum_i^n y_i/n$ viewed as a vector is the projection of $\mathbf{y}^T = [y_1 \ \cdots \ y_n]$ onto $\mathbf{1}_n$.
4. Using a programming language,
   a. Build a $N \times p$ matrix $\mathbf{X}$ filled with random variates drawn from the uniform distribution on the $[0, 1]$ interval. Set $N = 10000$ and $p = 5$.
   b. Scale each column of $\mathbf{X}$ to have unit length.
   c. Compute $\mathbf{X}^T\mathbf{X}$ and extract the largest and smallest *off-diagonal* elements. Compute the Frobenius norm of $\mathbf{X}^T\mathbf{X}$.
   d. Build a for loop and repeat steps a, b, c but increase p by 20 each time until $p = 2000$. Save the Frobenius norm, p, and the smallest and largest off-diagonal elements from each iteration.
   e. Plot p versus the minimums and maximums (on a single figure).
   f. Plot p versus Frobenius norms.

Chapter 2
Data Reduction

The topic of this chapter is very broad and the subject of much research. The oldest and most broadly applicable of data reduction methods is eigenvector analysis.

2.1 Introduction

Suppose that $\mathbf{X}$ is an $n \times p$ matrix constructed from n observations, each of which is a vector of length p. We have two and sometimes three goals: reducing the number of columns (or variables) from p to a smaller number, and understanding the relationships between the p variables that generated the observation vectors. Thirdly, relationships among the n observations sometimes are of interest.[1] These objectives are commonly pursued by reducing the dimensionality of $\mathbf{X}$. We turn now that that task.

The dimension of the data may be reduced using eigenvector techniques. Some loss of information is inevitable and so a primary theme is retaining as much information as possible while reducing the dimensionality. As mentioned above, the focus is on eigenvector techniques.

For example, suppose that $q << p$ and that

$$\underset{n \times q}{\mathbf{V}} = [\mathbf{v}_1 \ \mathbf{v}_2 \ \cdots \ \mathbf{v}_q]$$

contains most of the information contained in $\mathbf{X}$. In particular, it may be that $\mathbf{v}_1$ and $\mathbf{v}_2$ reflect two different and important latent sources of variation among the values in $\mathbf{x}_1, \ldots, \mathbf{x}_n$.[2] Plotting $\mathbf{v}_1$ versus $\mathbf{v}_2$ may reveal some

[1] A common objective is to form groups of observations that are alike within group and different between groups.

[2] For example, if $\mathbf{X}$ consists of measurements on physical attributes (height, weight, et cetera) of children, then age and gender are sources of variation in the values contained in the matrix $\mathbf{X}$.

aspect of the origins $\mathbf{X}$. Similarly, plotting or otherwise examining the linear transformations of the $n \times p$ matrix to a set of p-length vectors $\mathbf{X}\mathbf{v}_1$, $\mathbf{X}\mathbf{v}_2, \ldots, \mathbf{X}\mathbf{v}_q$ may also shed light onto the origins of $\mathbf{X}$.[3]

There's a variety of ways to develop the mathematical basis for eigenvector analysis. We'll take the spectral decomposition approach.

2.2 The spectral decomposition of a symmetric matrix

We approach the problem of reducing the dimensionality of a $n \times p$ data matrix $\mathbf{X}$ by attempting to find a lower dimensional approximation of $\mathbf{X}$ which may then be easier to analyze. We start not with $\mathbf{X}$ but instead a symmetric matrix $p \times p$ computed from $\mathbf{X}$. In particular, this matrix will be either a moment matrix (denoted as $\mathbf{M}$ below), a covariance matrix $\mathbf{S}$, or the Pearson product-moment (correlation) matrix $\mathbf{R}$, all of which are computed from $\mathbf{X}$. Each of these matrices can be expressed as both a sum over n outer products computed from the rows of $\mathbf{X}$, or as a product of an $n \times p$ matrix with itself. Specifically, the matrices and the two computational variations are

$$\begin{aligned} \mathbf{M} &= n^{-1} \sum_{i=1}^{n} \underset{p\times 1}{\mathbf{x}_{\cdot i}^T} \underset{1\times p}{\mathbf{x}_{\cdot i}} = \mathbf{X}^T\mathbf{X}, \\ \mathbf{S} &= n^{-1} \sum_{i=1}^{n} \underset{p\times 1}{(\mathbf{x}_{\cdot i} - \overline{\mathbf{x}})^T} \underset{1\times p}{(\mathbf{x}_{\cdot i} - \overline{\mathbf{x}})} = n^{-1}(\mathbf{X} - \mathbf{j}\overline{\mathbf{x}})^T(\mathbf{X} - \mathbf{j}\overline{\mathbf{x}}), \end{aligned} \tag{2.1}$$

where $\overline{\mathbf{x}}$ is the p-length row vector of means computed from the p columns of $\mathbf{X}$, $\mathbf{j}$ is an n-vector of ones (and therefore, a column vector), and

$$\mathbf{R} = n^{-1} \sum_{i=1}^{n} \underset{p\times 1}{\mathbf{z}_{\cdot i}^T} \underset{1\times p}{\mathbf{z}_{\cdot i}} = \mathbf{Z}^T\mathbf{Z},$$

where

$$\underset{1\times p}{\mathbf{z}_{\cdot i}} = [s_1^{-1}(x_{i,1} - \overline{x}_1) \;\; s_2^{-1}(x_{i,2} - \overline{x}_2) \;\; \cdots \;\; s_p^{-1}(x_{i,p} - \overline{x}_p)],$$

and s_j is the sample standard deviation of the jth variable.[4] The term n^{-1} that scales each matrix is not particularly important (except for $\mathbf{R}$) and may be omitted or modified with no meaningful consequences on the dimension reduction results.

[3] Each of the vectors $\mathbf{v}_j$ defines a linear transformation that may be applied to the rows of $\mathbf{X}$.

[4] $\mathbf{z}_i$ is the vector of standardized values of $\mathbf{x}_i$.

The *spectral decomposition* of a symmetric $p \times p$ matrix $\mathbf{A}$ expresses $\mathbf{A}$ as a sum of $p \times p$ matrices

$$\mathbf{A} = \sum_{i=1}^{p} \mathbf{A}_i, \tag{2.2}$$

where

$$\underset{p\times p}{\mathbf{A}_i} = \lambda_i \underset{p\times 1}{\mathbf{v}_i}\ \underset{1\times p}{\mathbf{v}_i^T}. \tag{2.3}$$

The matrix $\mathbf{A}_i$ is computed from a scalar *eigenvalue* $\lambda_i \geq 0$ and an *eigenvector* $\mathbf{v}_i$. Much of what drives the eigenvector analysis is the fact that $\mathbf{A}_i$ is the outer product of $\mathbf{v}_i$ with itself multiplied by the scalar λ_i. The rank of $\mathbf{A}_i$ is the same as $\mathbf{v}_i$ and therefore, is 1. Much of what we learn about $\mathbf{A}$ through the eigenvector analysis is realized through the eigenvectors and eigenvalues rather than the outer products.

Usually, the eigenvectors are scaled to have length 1 (hence, $\mathbf{v}_i^T\mathbf{v}_i = 1$) and the $\mathbf{A}_i$'s are roughly the same magnitude each having been constructed from a vectors with the same norm. If so, then the eigenvalues represent the importance of each eigenvector pair towards constructing $\mathbf{A}$ in the spectral decomposition (equations 2.2 and 2.3). For simplicity, assume that no two eigenvalues are equal, and that the eigenvalues are arranged in order of magnitude:

$$0 \leq \lambda_p < \lambda_{p-1} < \cdots < \lambda_2 < \lambda_1.$$

We say that $\mathbf{v}_1$ is the dominant eigenvector because λ_1 is largest and $\mathbf{A}_1$ has the greatest contribution to the sum in equation (2.2).

Since $\mathbf{A}_i$ is built from a single eigenvector pair $(\lambda_i, \mathbf{v}_i)$, $\mathbf{A}_i$ has a rank[5] of 1 whereas the rank of $\mathbf{A}$ may be as large as p, much larger compared to the rank of the component matrices $\mathbf{A}_1, \ldots, \mathbf{A}_p$. What is surprising about the spectral decomposition is that $\mathbf{A}_1$ is often a good approximation of $\mathbf{A}$. Thus, much of the information contained in $\mathbf{A}$ is present in $\mathbf{A}_1$ and therefore, in $\mathbf{v}_1$ because $\mathbf{A}_1$ is a simple function of $\mathbf{v}_1$. Since $\mathbf{v}_1$ is a p-vector, it's much easier to analyze than the original $p \times p$ matrix $\mathbf{A}$ and also $\mathbf{X}$ from which $\mathbf{A}$ was constructed. Even if $q > 1$ eigenvector pairs[6] are necessary to get a good approximation of $\mathbf{A}$, the analysis of the first few eigenvector pairs is often productive.[7,8]

[5] The rank of a matrix is the number of linearly independent column vectors in the matrix. Heuristically, the rank is a measure of the information content of a matrix.

[6] The ith eigenvector pair is $(\lambda_i, \mathbf{v}_i)$

[7] The eigenvalues provide valuable information about the intrinsic dimensionality of $\mathbf{A}$.

[8] If q eigenvector pairs are necessary to get a good approximation, then we may write

$$\mathbf{A} \approx \sum_{i=1}^{q} \mathbf{A}_i. \tag{2.4}$$

2.3 Derivation of the Dominant Eigenvector

Recall that the inner product of two vectors $\mathbf{x}$ and $\mathbf{v}$ is

$$\mathbf{x}^T\mathbf{v} = ||\mathbf{x}||||\mathbf{v}|| \cos(\mathbf{x}, \mathbf{v}), \tag{2.5}$$

where $\cos(\mathbf{x}, \mathbf{v})$ is the cosine of the angle between the vectors $\mathbf{x}$ and $\mathbf{v}$ and $||\mathbf{v}|| = (\sum_j v_j^2)^{1/2}$ is the L_2 or Euclidean norm of $\mathbf{v}$. If the angle between the vectors is small, the vectors point in the nearly the same direction and the cosine of the angle will be close to 1. We will say that the vectors are nearly *coincident*. If the data consist of pairs $(x_j, v_j), j = 1, 2, \ldots, n$ and $\mathbf{x}$ and $\mathbf{v}$ are constructed from these data[9], then the correlation[10] between $\mathbf{x}$ and $\mathbf{v}$, denoted by $r(\mathbf{x}, \mathbf{v})$, equals $\cos(\mathbf{x}, \mathbf{v})$. In particular,

$$\frac{\mathbf{x}^T\mathbf{v}}{||\mathbf{x}||||\mathbf{v}||} = \cos(\mathbf{x}, \mathbf{v}) = r(\mathbf{x}, \mathbf{v}). \tag{2.6}$$

Informally speaking, the dominant eigenvector is the vector that is most coincident with (all) of the column vectors $\mathbf{x}_{\cdot 1}, \ldots, \mathbf{x}_{\cdot p}$ comprising $\mathbf{X}$. Often, it's easy to gain some understanding of $\mathbf{X}$ by studying the dominant eigenvector. Other eigenvectors may be extracted from the data matrix as well, and these often reveal more (and different) information contained in the data.

Suppose that the data matrix $\mathbf{X}$ has been transformed to a product such as one of the examples given in equations (2.1). To be specific, consider the product-moment matrix $\mathbf{A} = \mathbf{X}^T\mathbf{X}$. The objective is to find a vector $\mathbf{v}_1$ with norm $||\mathbf{v}_1|| = 1$ that is maximally coincident with $\mathbf{X}$ in the following sense:

$$\mathbf{v}_1 = \underset{||\mathbf{v}||=1}{\arg\max}\ \mathbf{v}^T\mathbf{A}\mathbf{v}. \tag{2.7}$$

Equation (2.7) states that $\mathbf{v}_1$ is the vector of norm 1 that produces the maximum possible value of the *quadratic product* $\mathbf{v}^T\mathbf{A}\mathbf{v}$. The motivation for maximizing the quadratic product can be understood by expanding the product:

$$\mathbf{v}^T\mathbf{A}\mathbf{v} = \mathbf{v}^T\mathbf{X}^T\mathbf{X}\mathbf{v}. \tag{2.8}$$

Note that $\mathbf{X}\mathbf{v}$ is a n vector of inner products created from the row vectors of $\mathbf{X}$ and the vector $\mathbf{v}$. Therefore,

$$\underset{n\times 1}{\mathbf{X}\mathbf{v}} = \begin{bmatrix} \underset{1\times p}{\mathbf{x}_{1\cdot}}\ \underset{p\times 1}{\mathbf{v}} \\ \vdots \\ \underset{1\times p}{\mathbf{x}_{n\cdot}}\ \underset{p\times 1}{\mathbf{v}} \end{bmatrix}, \tag{2.9}$$

[9] Hence, $\mathbf{x} = [x_1\ \cdots\ x_n]^T$ and $\mathbf{v} = [v_1\ \cdots\ v_n]^T$.

[10] Specifically, Pearson's correlation coefficient.

Because $||\mathbf{v}|| = 1$, and $\mathbf{x}_{i\cdot}\mathbf{v} = ||\mathbf{x}_{i\cdot}|| \cos(\mathbf{x}_{i\cdot}, \mathbf{v})$, we have

$$\begin{aligned} \mathbf{v}^T\mathbf{A}\mathbf{v} =& (\mathbf{X}\mathbf{v})^T\mathbf{X}\mathbf{v} \\ =& \sum_{i=1}^{n} (\mathbf{x}_{i\cdot}\mathbf{v})^2 \\ =& \sum_{i=1}^{n} ||\mathbf{x}_{i\cdot}||^2 \cos^2(\mathbf{x}_{i\cdot}, \mathbf{v}). \end{aligned} \tag{2.10}$$

The vector $\mathbf{v}$ does not affect the row norms $||\mathbf{x}_{1\cdot}||, \ldots, ||\mathbf{x}_{n\cdot}||$, and so the task posed by equation (2.7) reduces to finding the vector $\mathbf{v}$ that maximizes the sum of the squared cosines, or equivalently, the squared correlations between the row vectors and $\mathbf{v}$.[11]

The maximized quadratic function (shown in equations 2.10) is the first eigenvalue λ_1. Specifically,

$$\lambda_1 = \mathbf{v}_1^T\mathbf{A}\mathbf{v}_1 = \mathbf{v}_1^T\mathbf{X}^T\mathbf{X}\mathbf{v}_1, \tag{2.11}$$

and it is commonly referred to as the dominant eigenvalue and sometimes as the Rayleigh quotient. Returning to equations (2.2 and 2.3), and summarizing,

$$\underset{p\times p}{\mathbf{A}_1} = \lambda_1 \underset{p\times 1}{\mathbf{v}_1} \underset{1\times p}{\mathbf{v}_1^T}$$

is the best possible approximation of $\mathbf{A}$.[12]

Much can be learned from the coefficients comprising $\mathbf{v}$ (also known as *loadings*) because $v_{i,1}$ determines how the ith variable (corresponding to the ith column of $\mathbf{X}$) contributes to the best approximation of $\mathbf{A}$. If the magnitude $v_{i,1}$ is large in magnitude, then the ith variable is an important determinant of process that generated the values contained in $\mathbf{X}$. Since $||\mathbf{v}|| = 1$, the coefficients are restricted in magnitude and the question of whether a coefficient is small or large in magnitude is relatively easy to resolve, though it does depend in part on the number of elements in the p-element vector $\mathbf{v}_1$.

Note that we may create $\mathbf{A}$ in such a way that *observations* are the principal subject of analysis (rather than variables). For example, we may create $\mathbf{A}$ by computing $\underset{n\times n}{\mathbf{A}} = \mathbf{X}\mathbf{X}^T$. This approach is particularly useful when analyzing multivariate time series since the columns of $\mathbf{X}$ can be set up so that each column contains the observations obtained from the p variables at a particular time. Presumably the columns would be arranged in chronological order.

[11] There's an issue of direction since large values of $\cos^2(\mathbf{x}_i, \mathbf{v})$ reflect only the strength of the directional coincidence between vectors regardless of sign. However, this issue turns out to be minor. The main point is that the objective of eigenvector analysis is to find a vector $\mathbf{v}_1$ that is strongly correlated (positively or negatively) to the observations comprising $\mathbf{X}$.

[12] Specifically, the Frobenius norm of $\mathbf{A}_1 - \mathbf{A}$ cannot be reduced by a different choice of $\mathbf{v}$.

In most situations, we continue beyond extracting the dominant eigenvector pair by extracting a second eigenvector pair. The eigenvector pair $(\lambda_2, \mathbf{v}_2)$ is maximally coincident of the rows of the residual matrix $\mathbf{A} - \mathbf{A}_1$. Furthermore, $\mathbf{v}_2$ is orthogonal to $\mathbf{v}_1$—in other words, it's correlation with $\mathbf{v}_1$ is zero. Mathematically, the second eigenvector pair can be defined by

$$\mathbf{v}_2 = \underset{||\mathbf{v}||=1, \mathbf{v}_1^T\mathbf{v}=0}{\arg\max} \mathbf{v}^T(\mathbf{A} - \mathbf{A}_1)\mathbf{v}. \tag{2.12}$$

If we set

$$\underset{p\times p}{\mathbf{A}_2} = \lambda_2 \mathbf{v}_2 \mathbf{v}_2^T,$$

then the residual matrix $\mathbf{A}_3 = \mathbf{A} - \mathbf{A}_1 - \mathbf{A}_2$ may also be processed the same way to yield the third eigenvector pair $(\lambda_3, \mathbf{v}_3)$. While the p eigenvector pairs are correctly defined and computed by this iterative scheme, the calculations of the eigenvector pairs is usually not iterative, but instead accomplished a more sophisticated algorithm that computes all eigenvector pairs nearly simultaneously.[13] Furthermore, the complete set of eigenvector pairs are usually defined without recourse to this iterative scheme.

2.4 Principal Components

Let's suppose that all p eigenvector pairs have been extracted from a symmetric $p \times p$ matrix and ordered from largest to smallest. Mathematically, the product of the original matrix ($\mathbf{X}$ or $\mathbf{Z}$) with the ith eigenvector is a *projection* of the data onto the ith eigenvector axis. Consider a specific row $\mathbf{x}_i$ (it's presumed that $\mathbf{x}_i$ is a p-vector of observations). Equation (2.5) shows that the projection of $\mathbf{x}_i$ onto a eigenvector $\mathbf{v}$ is the cosine of the angle between $\mathbf{x}_i$ and $\mathbf{v}$ scaled by $||\mathbf{x}_i||$, i.e.,

$$\begin{aligned} \text{proj}_{\mathbf{v}}(\mathbf{x}_i) &= \underset{1\times pp\times 1}{\mathbf{x}_i\mathbf{v}} \\ &= ||\mathbf{x}_i|| \cos(\mathbf{x}_i, \mathbf{v}). \end{aligned}$$

If we view $\mathbf{v}$ as a describing a new axis, then $\text{proj}_{\mathbf{v}}(\mathbf{x}_i)$ is the position on $\mathbf{v}$ at which a line segment connecting the endpoint of $\mathbf{x}_i$ to $\mathbf{v}$ intersect. The connecting line is perpendicular to the eigenvector axis.[14] The line segment along $\mathbf{v}$ from the origin to the intersection is called the *projection* of the observation vector onto the eigenvector. To illustrate, figure 1 shows the projections of two vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ onto an axis $\mathbf{v}$. The projection of the

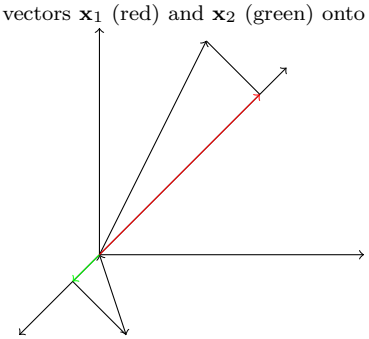
[13] However, if the matrices of interest are very large, the computational effort may be insurmountable and the iterative approach might be a to the computational challenge.

[14] Therefore, the connecting line segment is the shortest possible segment connecting $\mathbf{v}$ and $\mathbf{x}_i$.

vector $\mathbf{x}_1$ is substantially greater in length than the projection of $\mathbf{x}_2$ even after accounting for the differences in length of $\mathbf{x}_1$ and $\mathbf{x}_2$.

If the vectors comprising $\mathbf{X}$ are in $\mathbb{R}^2$ (therefore, $p = 2$), and two eigenvectors are extracted from $\mathbf{M} = \mathbf{X}^T\mathbf{X}$, then the eigenvectors are often thought of as a pair of axes obtained by rotating the usual, or canonical, horizontal and vertical axes. Both eigenvector axes are rotated to the same degree since orthogonality of the original axes is automatically maintained by the condition that $\mathbf{v}_1^T\mathbf{v}_2 = 0$. The degree of rotation is determined by the eigenvector objective of maximizing the sum of the projection lengths onto the axes. The

Fig. 1 Projection of the vectors $\mathbf{x}_1$ (red) and $\mathbf{x}_2$ (green) onto the vector $\mathbf{v}$.



first eigenvector provides an axis with an interesting property: the sum of projections lengths of the rows of $\mathbf{X}$ (or $\mathbf{Z}$) onto the axis are as large as possible because the sum of the squared projection lengths is the inner product of the projections ($\mathbf{X}\mathbf{v}_1$) with itself, namely,

$$\begin{aligned}(\mathbf{X}\mathbf{v}_1)^T\mathbf{X}\mathbf{v}_1 =& \mathbf{v}_1^T\mathbf{X}^T\mathbf{X}\mathbf{v}_1 \\ =& \mathbf{v}_1^T\mathbf{M}\mathbf{v}_1 \\ =& \lambda_1.\end{aligned} \tag{2.13}$$

In this calculation, we've set $\mathbf{M} = \mathbf{X}^T\mathbf{X}$ and have used the facts that $\mathbf{M}\mathbf{v}_1 = \lambda\mathbf{v}_1$ and $\mathbf{v}_1^T\mathbf{v}_1 = 1$. No other vector with a norm of 1 will produce a larger inner product and larger eigenvalue. In applications, there's a great deal of interest in the *first principal component* $\mathbf{X}\mathbf{v}_1$ since it is determined by a straightforward linear function of the rows of the data matrix (straightforward is relative of course). The coefficients of $\mathbf{v}_1$ are sometimes called *loading* because the ith coefficient v_i describes the weight assigned to the ith variable in the mapping of an observation vector $\mathbf{x}$ onto the eigenvector axis determined by $\mathbf{v}_1$. Note that if several rows, say $\mathbf{x}_{\cdot i}, \mathbf{x}_{\cdot j}, \ldots, \mathbf{x}_{\cdot k}$ have similar projections onto $\mathbf{v}_1$, then the corresponding observations are alike with respect to the dominant eigen-axis. Furthermore, if the observation vec-

tors are recorded on specific units, say, individuals, then there's evidence that the units are alike, perhaps belonging to a single sub-population.

Other principal components are also of interest. The relative importance of each eigenvector pair is measured by the eigenvalue, and so if λ_i is much smaller than the preceding eigenvalues, then $\mathbf{A}_i$ does not explain much of $\mathbf{A} = \mathbf{X}^T\mathbf{X}$ and the eigenvector pair $(\lambda_i, \mathbf{v}_i)$ is not of much importance. A dimension reduction analysis that focuses mapping a p-dimensional data matrix to a lower q-dimensional representation, say,

$$\underset{n\times q}{\mathbf{P}} = \underset{n\times p}{\mathbf{X}}\underset{p\times q}{[\mathbf{v}_1\ \mathbf{v}_2\ \cdots\ \mathbf{v}_q]} \tag{2.14}$$

is often called *principal components analysis*.[15] To illustrate, figure 2 shows the first and second eigenvector axes computed from a covariance matrix $\mathbf{S}$. The data ($n = 1000$), were sampled from a multivariate normal distribution and produced the eigenvalues $\lambda_1 = 2.61$ and $\lambda_2 = .23$. Since $\lambda_1 >> \lambda_2$, the first eigenvector axis accounts for the great majority of variation ($\sim 92\%$, since $2.61/(2.61 + .23) = .92$) in the sample.[16] If one imagines the data

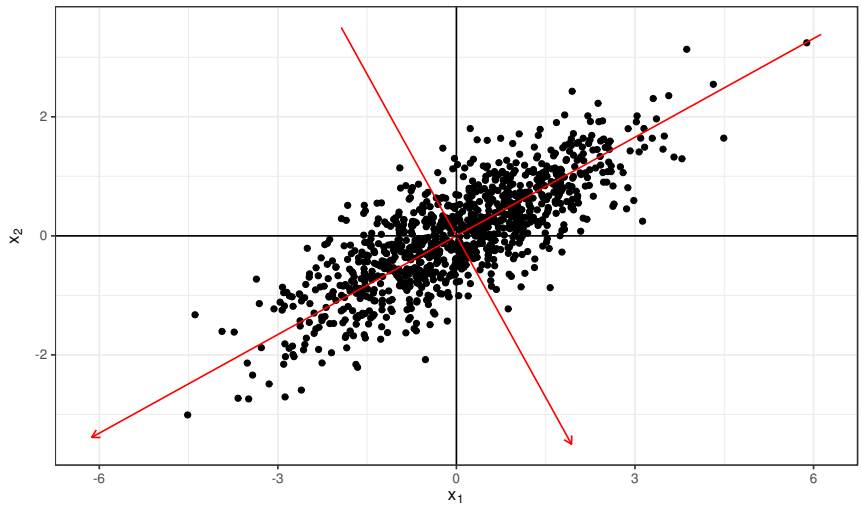


Fig. 2 Eigenvector axes computed from the covariance matrix of a sample of multivariate normal observations.

points plotted on the red (eigenvector) axes, the points would describe an

[15] For example, there's a popular built-in `R` function named `princomp` that creates a number of objects related to principal components analysis.

[16] This claim is consistent with the Pearson's correlation coefficient measuring association between the two columns ($r = .79$).

ellipse and the cloud points would not manifest any correlation. In fact, we can show mathematically that the new points are uncorrelated. Because the eigenvector pairs have been extracted from the variance matrix $\mathbf{S}$ (equation 2.1), let $\mathbf{X}_c = n^{-1/2}(\mathbf{X} - \mathbf{j}\overline{\mathbf{x}}^T)$ denote the centered data matrix. Also let $\mathbf{w} = \mathbf{X}_c\mathbf{v}_1$, and $\mathbf{u} = \mathbf{X}_c\mathbf{v}_2$ denote the first and second principal component score vectors. The inner product of $\mathbf{w}$ and $\mathbf{u}$ is proportional to the correlation $r(\mathbf{w}, \mathbf{u})$ between the first and second principal component scores, and the inner product is

$$\mathbf{w}^T\mathbf{u} = \mathbf{v}_1^T\mathbf{S}\mathbf{v}_2 = \lambda_2\mathbf{v}_1^T\mathbf{v}_2 = 0. \tag{2.15}$$

Hence, the principal component scores are uncorrelated.

Before continuing with the mathematical development, a practical example is presented in the next section.

2.5 Eigenvector Analysis of Stock Prices

We consider a data set consisting of $n = 64$ stocks. Each stock provides a time series consisting of daily closing price spanning $p = 2009$ consecutive trading days. Thus, the data matrix consists of high-dimensional vectors ($p = 2009$), and consequently, analyzing the patterns of variation in the data and relationships among observations (based on the p variables) is difficult. Eigenvector analysis provides a solution by projecting the p-dimensional observations onto the first and dominant eigenvector axis. Then, it's easy to order the observations according the values of the projections of each vector onto the axis and compare a selected handful of original observations. In principle, one would be interested in identifying a group of stocks that are alike with respect to the dominant eigenvector axis as each stock in the group may be useful for predicting the other stocks in the group. One can go farther by projecting the data onto a few dominant eigenvector axes thereby extracting somewhat more information from the original matrix.

Herein, our interest is the same as in cluster analysis—we wish to discriminate among stocks based on the series of closing prices observed on each stock. We're particularly interested in identifying similar and dissimilar stocks since a set of stocks with trend similar to a target stock are likely to be useful for predicting the future values of the target. In this analysis, we'll not form clusters and so the analysis is somewhat more exploratory in nature than cluster analysis.[17] Specifically, it's hoped that the results of the eigenvector analysis will reveal specific patterns of trends. Furthermore, we hope to find groups of stocks that will serve as initial clusters because the

[17] The results of this analysis may provide a useful initial set of clusters to a k-means algorithm.

result of the k-means algorithm depends to an unknown extent on the initial configuration of clusters.[18]

The series have been scaled so that the mean and standard deviation of each stock series is 0 and 1, respectively.[19] The matrix of scaled values is denoted as

$$\underset{64\times 2009}{\mathbf{Z}} = \begin{bmatrix} \mathbf{z}_{1\cdot} \\ \mathbf{z}_{2\cdot} \\ \vdots \\ \mathbf{z}_{n\cdot} \end{bmatrix}, \tag{2.16}$$

where $\mathbf{z}_{i\cdot}$ is a series of $p = 2009$ observations on the ith stock.

The first preliminary step is to transform $\mathbf{Z}$ to a $p \times p$ scaled moment matrix $\mathbf{M} = (n-1)^{-1}\mathbf{Z}\mathbf{Z}^T$.[20] The elements of $\mathbf{Z}\mathbf{Z}^T$ are inner products between series, and so each element of the matrix reflects the similarity (or coincidence) between two series. Because the series have been scaled to have a standard deviation of 1, the ith series has squared norm $(\sum_k^n z_{i,k}^2) = n - 1$, for every i. Consequently, the i, jth element of $\mathbf{M}$ is the Pearson's correlation coefficient between series i and j.[21] The dominant eigenvector $\mathbf{v}_1$ is the p-element vector with $||\mathbf{v}_1|| = 1$ that maximizes $||\mathbf{M}\mathbf{v}_1||$. Consequently, we may say (perhaps informally) that $\mathbf{v}_1$ is maximally aligned with the rows of $\mathbf{M}$.

Figure 3 shows all $p = 64$ eigenvalues and the cumulative sum of the eigenvalues plotted against the eigenvalue label $1, 2, \ldots, 64$. The sum of all 64 eigenvalues is the rank of $\mathbf{X}$ (64 in this instance). We routinely interpret the cumulative sum $\sum_{i=1}^k \lambda_i$ as the proportion of total variation in the data accounted for by the first k eigenvector pairs. By this measure, 3 or 4 eigenvector pairs are adequate for constructing a low-dimensional approximation of the moment matrix. Further, λ_1 accounts for 62.8% of the variation from which we infer that there is a single, strong underlying process, probably manifested as a common trend, that is determining $\mathbf{A}$ and $\mathbf{X}$.

The coefficients forming the first eigenvector are plotted in Figure 5. The majority of the coefficients are roughly equal, suggesting that the first eigenvector is a manifestation of a common underlying signal because the contributions of each of the n stocks toward determining the first principal component are similar. Not all stocks exhibit the signal equally though because the coefficients are not all the same, and in fact, a few of the coefficients exhibit a different sign implying that the associated stocks are exhibiting a much

[18] A set of stocks that exhibit similar trends that are used collectively to forecast future closing prices probably will yield more accurate forecasts than a single stock used for the same purpose.

[19] The scaling of *rows* is differs from the usual scaling of *columns*. Scaling each variable (column) is logical if the variables are are measured in different units, but in this case, the units are the same—dollars.

[20] Alternatively, any of the transformations shown in equation (2.1) suffice.

[21] We scale the terms in $\mathbf{Z}\mathbf{Z}^T$ as each term is the inner product of n terms.

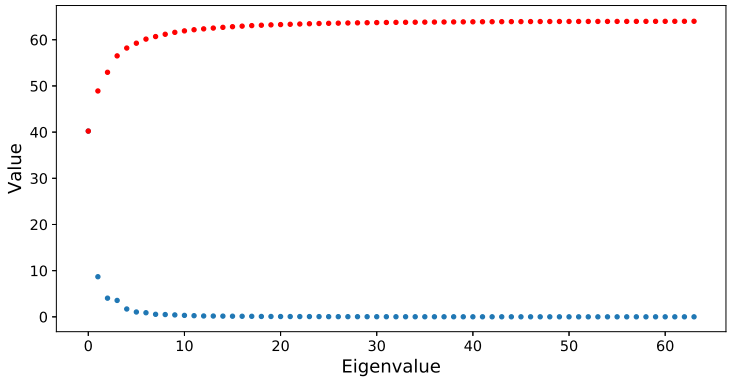


Fig. 3 Eigenvalues (blue) and the cumulative sum of eigenvalues (red) plotted against the ordered eigenvalue label (ordered according to largest eigenvalue to smallest eigenvalue). Note: the first cumulative sum (red) is plotted over the first eigenvalue (blue). The two terms are equal.

different signal. An examination of the first principal component, $\mathbf{Zv}_1$ plotted against date confirms this inference (Figure 6). This figure shows that the linear transformation of stock prices given by $\mathbf{Zv}_1$ declines consistently with date. The implication is that there is a strong signal in the data set that is driving prices upward (or downward).[22] The second eigenvector axis is unrelated to date. To ascertain whether these inferences are correct, we plot the 6 time series generated by the stocks with the largest and smallest associated first eigenvector coefficients in Figure 4. These are VVC, AOS, and BDX (smallest coefficients) and YGE, MT, and HHS (largest coefficients). Since the stocks with the smallest (and negative) coefficients exhibit positive upward trend, we conclude that the majority of stocks increased in price over the time span.

2.6 Properties of Eigenvectors

Previously, we described the eigenvector pairs as resulting from an algorithm that successively reduces the error associated with an approximation of the $p \times p$ symmetric matrix $\mathbf{A}$. In brief, the approximation is constructed as a sum of matrices, each of which is an outer product of an eigenvector $\mathbf{v}_i$ with itself and an eigenvalue λ_i:

[22] The eigenvector can be multiplied by -1 and it still is an eigenvector. However, the trend manifested by the first principal component scores will be upward instead of downward.

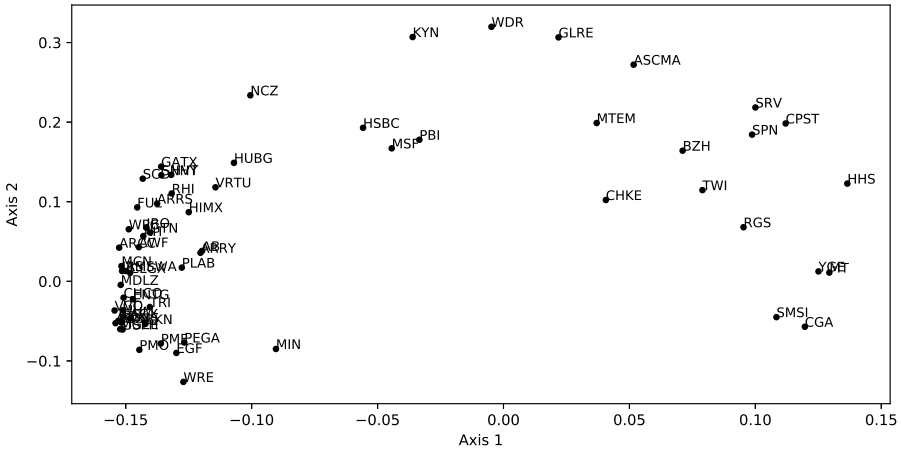


Fig. 4 Projections of each price series onto the first and second eigenvector axes. Series are identified by the stock symbol.

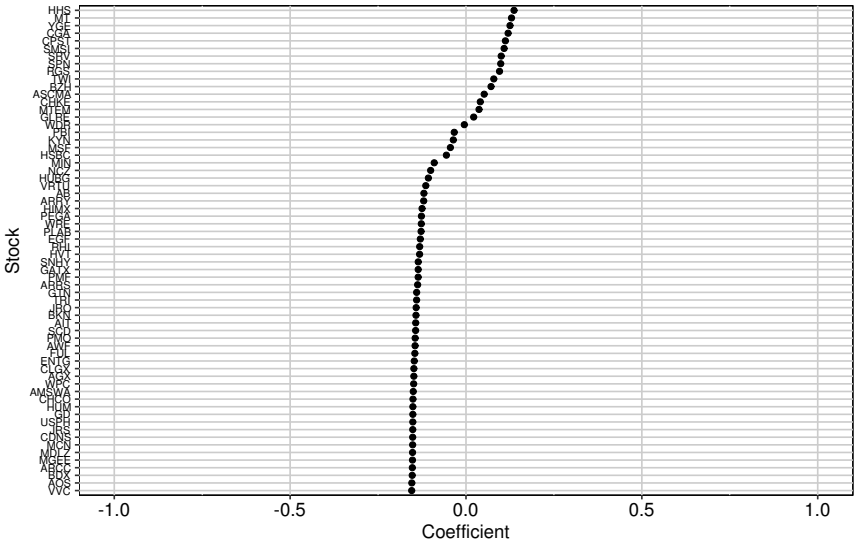


Fig. 5 Stock symbol plotted against first eigenvector coefficients. The x-axis spans the range of valid coefficients. Many coefficients are nearly the same value ($-.15$).

$$\mathbf{A} \approx \sum_{i=1}^{q} \lambda_i \mathbf{v}_i \mathbf{v}_i^T, \tag{2.17}$$

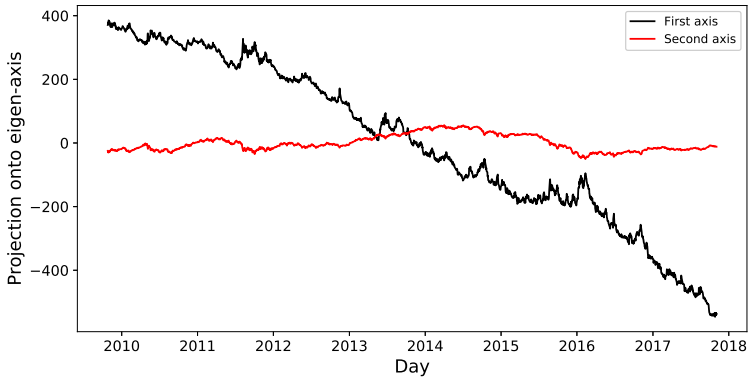


Fig. 6 First and second principal component scores ($\mathbf{Z}\mathbf{v}_1$ and $\mathbf{Z}\mathbf{v}_2$) plotted against date. The consistent downward trend with date for the first principal component scores imply that $\mathbf{v}_1$ reflects a common trend in stock prices between 2010 and 2018. There's no significant trend over time in the second principal component scores.

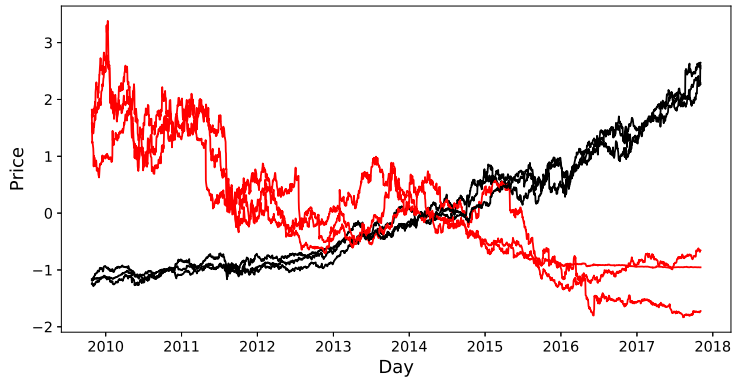


Fig. 7 Standardized stock series ($\mathbf{Z}_{\cdot j}$'s) corresponding to stocks with largest and smallest first eigenvector coefficients plotted against time. Red lines: YGE, MT, and HHS; black lines: VVC, AOS, and BDX.

where $1 \leq q < p$. There are p eigenvector pairs $(\lambda_i, \mathbf{v}_i), i = 1, 2, \ldots, p$, and the sum over all p reconstructs $\mathbf{A}$ without error. The eigenvectors $\mathbf{v}_1$, $\mathbf{v}_2$, …, $\mathbf{v}_p$ are constrained to be orthogonal (equation 2.12) and the sum over the complete set of eigenvector pairs is the *spectral decomposition* of $\mathbf{A}$.

Exercise 8 demonstrates that the spectral decomposition of $\mathbf{A}$ also can be written as a matrix product

$$\mathbf{V}\mathbf{\Lambda}\mathbf{V}^T = \sum_{i=1}^{p} \lambda_j \mathbf{v}_i \mathbf{v}_i^T, \tag{2.18}$$

where $\underset{p\times p}{\mathbf{V}} = [\mathbf{v}_1\ \mathbf{v}_2\ \cdots\ \mathbf{v}_p]$ is constructed from the eigenvectors and

$$\mathbf{\Lambda} = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_p \end{bmatrix}$$

is constructed from the eigenvalues.

A number of interesting results can be derived. A few are reviewed in the following discussion.

2.7 Mathematical Results Concerning Eigenvector Pairs

We'll start with a short calculation that leads to two alternative definitions of the eigenvectors and values (primarily to gain familiarity with working with eigenvector pairs rather than insight). Because the eigenvectors are orthogonal and are constrained to have Euclidean norm 1, $\mathbf{V}^T\mathbf{V} = \mathbf{I}_p$ where $\mathbf{I}_p$ is the $p \times p$ identity matrix. Let $\mathbf{v}_j$ denote the jth column of $\mathbf{V}$ (and the jth eigenvector). As usual, $\mathbf{v}_i$ denotes the ith eigenvector. Then, the projection of $\mathbf{A}$ onto the ith eigenvector is

$$\begin{aligned} \mathbf{A}\mathbf{v}_i &= \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T\mathbf{v}_i \\ &= \mathbf{V}\mathbf{\Lambda}\begin{bmatrix} \mathbf{v}_1^T\mathbf{v}_i \\ \vdots \\ \mathbf{v}_i^T\mathbf{v}_i \\ \vdots \\ \mathbf{v}_p^T\mathbf{v}_i \end{bmatrix} = \mathbf{V}\mathbf{\Lambda}\begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} = \mathbf{V}\begin{bmatrix} 0 \\ \vdots \\ \lambda_i \\ \vdots \\ 0 \end{bmatrix} \\ &= \lambda_i \mathbf{v}_i. \end{aligned} \tag{2.19}$$

The result that $\mathbf{A}\mathbf{v}_i = \lambda_i\mathbf{v}_i$ is commonly used as the starting point for defining the eigenvectors pairs. For example, the eigenvectors pairs are defined so: $(\lambda_1, \mathbf{v}_1)$ is a pair that satisfies $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ and $(\lambda_2, \mathbf{v}_2)$ is a pair that satisfies $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ given that $\mathbf{v}_1^T\mathbf{v}_2 = 0$, and so on. Furthermore, this result involving the single eigenvector $\mathbf{v}_i$ can be extended to the matrix of eigenvectors:

$$\begin{aligned}\mathbf{AV} &= \mathbf{A}[\mathbf{v}_1 \ \cdots \ \mathbf{v}_p] \\ &= [\mathbf{A}\mathbf{v}_1 \ \cdots \ \mathbf{A}\mathbf{v}_p] \\ &= [\lambda_1\mathbf{v}_1 \ \cdots \ \lambda_p\mathbf{v}_p] \\ &= \mathbf{\Lambda V}.\end{aligned} \tag{2.20}$$

Sometimes the eigenvectors and eigenvalues are defined as the terms forming an orthogonal matrix $\mathbf{V}$ and a diagonal matrix $\mathbf{\Lambda}$ satisfying

$$\mathbf{AV} = \mathbf{\Lambda V}. \tag{2.21}$$

Some additional results are:

1. Since each of the eigenvectors are orthogonal, $\mathbf{V}$ is composed of p linearly independent vectors (exercise 4 asks the reader to prove this claim), Because of linear independence of the columns, $\mathbf{V}$ has an inverse $\mathbf{V}^{-1}$. Thus, $\mathbf{V}\mathbf{V}^{-1} = \mathbf{I}_p$, and
$$\begin{aligned}\mathbf{V}^T\mathbf{V}\mathbf{V}^{-1} &= \mathbf{V}^T \\ \Rightarrow \mathbf{V}^{-1} &= \mathbf{V}^T.\end{aligned} \tag{2.22}$$
In other words, the inverse of $\mathbf{V}$ is $\mathbf{V}^T$ and the inverse of $\mathbf{V}^T$ is $\mathbf{V}$.
2. If $\mathbf{A}$ is invertible, then the inverse of $\mathbf{A}$ is $\mathbf{V}\mathbf{\Lambda}^{-1}\mathbf{V}^T$. This statement implies that all of the eigenvalues are non-zero. Exercise 6 asks for a proof. It also implies that the eigenvectors of $\mathbf{A}^{-1}$ are the eigenvectors of $\mathbf{A}$ and the eigenvalues of $\mathbf{A}^{-1}$ are the reciprocals of the eigenvalues of $\mathbf{A}$.
3. The *trace* of square matrix is the sum of the diagonal elements. The trace of a square matrix is the sum of the eigenvalues. Also, the sum of the eigenvalues of correlation matrix is the number of rows of the correlation matrix (since the diagonal consists of ones). Similarly, the sum of the eigenvalues of a covariance matrix is the sum of the sample variances, and
$$\sum_{i=1}^{p} \lambda_i = \sum_{i=1}^{p} s_i^2. \tag{2.23}$$
4. The number of non-zero eigenvalues is the rank of $\mathbf{A} = \mathbf{X}^T\mathbf{X}$ and the rank of $\mathbf{X}$.

2.8 The Power Method

The power method is an algorithm for computing eigenvector pairs of a large matrix. For small matrices, it is either slow or imprecise and so it is rarely used. When matrices are excessively large, the usual methods built into the standard linear algebra packages cannot be used and a recourse is the power method. Typically, the power method is used to compute the first few and

hence, most important eigenvector pairs rather than the complete set of pairs. It's not complicated and is readily programmed in `Python` or `R`.

An overview begins by setting $\mathbf{A}$ to be the matrix from which the first few eigenvector pairs are to be extracted. The power method repeatedly multiplies $\mathbf{A}$ by a p-vector yielding another p-vector which is then scaled to have norm 1. The next iteration repeats the two steps by multiplying $\mathbf{A}$ and the previously computed vector. The algorithm will yield an approximation of the dominant eigenvector pair $(\lambda_1, \mathbf{v}_1)$ after a number of iterations.[23] To get the second eigenvector pair, the algorithm is applied again to the residual matrix $\mathbf{R}_1 = \mathbf{A} - \lambda_1 \mathbf{v}_1 \mathbf{v}_1^T$.

It may be helpful to recall the following facts before developing the algorithm in detail.

1. Any linearly independent set of p p-length vectors will serve as a basis for $\mathbb{R}^p$. For example, the *standard* basis is

$$\mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \cdots, \mathbf{e}_p = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}.$$

   Given any $\mathbf{x} \in \mathbb{R}^p$, there exists a set of real numbers $\alpha_1, \alpha_2, \ldots, \alpha_p$ such that

$$\mathbf{x} = \alpha_1 \mathbf{e}_1 + \alpha_2 \mathbf{e}_2 + \cdots + \alpha_p \mathbf{e}_p,$$

   because $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_p$ form a basis for $\mathbb{R}^p$. The coefficients are $\alpha_1 = x_1, \alpha_2 = x_2, \ldots, \alpha_p = x_p$.
2. A set of p p-length eigenvectors are linearly independent and therefore, is a basis for $\mathbb{R}^p$.
3. Statement 2 implies that every vector $\mathbf{x} \in \mathbb{R}^p$ can be written as a linear combination of the p-length eigenvectors $\mathbf{v}_1$, $\mathbf{v}_2$, …, $\mathbf{v}_p$ extracted from a square $p \times p$ matrix. Said another way, given any $\mathbf{x} \in \mathbb{R}^p$, there exists a set of real numbers $\alpha_1, \alpha_2, \ldots, \alpha_p$ such that

$$\mathbf{x} = \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \cdots + \alpha_p \mathbf{v}_p = \sum_i \alpha_i \mathbf{v}_i. \tag{2.24}$$

4. Let $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_p\}$ denote the eigenvectors of the $p \times p$ matrix $\mathbf{A}$. Note that $\mathbf{A}(\lambda_1 \mathbf{v_1}) = \lambda_1 \mathbf{A}\mathbf{v}_1 = \lambda_1^2 \mathbf{v}_1$. Suppose that both sides of the equation $\mathbf{A}\mathbf{v}_1 = \lambda_1 \mathbf{v}_1$ are pre-multiplied (on the left) $k-1$ times by $\mathbf{A}$. The result is

$$\begin{aligned} \underbrace{\mathbf{A} \times \cdots \times \mathbf{A}}_{k \text{ terms}} \mathbf{v}_1 &= \lambda_1 \underbrace{\mathbf{A} \times \cdots \times \mathbf{A}}_{k \text{ terms}} \mathbf{v}_1 \\ \Rightarrow \mathbf{A}^k \mathbf{v}_1 &= \lambda^k \mathbf{v}_1. \end{aligned} \tag{2.25}$$

[23] The approximation error can be driven to the limits of machine precision with enough iterations.

This calculation establishes that $(\lambda^k, \mathbf{v}_1)$ is an eigenvector pair of $\mathbf{A}^k$. It's also true that $\mathbf{A}^k$ has the same eigenvectors as $\mathbf{A}$ and that the eigenvalues of $\mathbf{A}^k$ can be obtained from the eigenvalues of $\mathbf{A}$, and vice versa. One may also show that λ_j^k is the jth largest eigenvalue of $\mathbf{A}^k$, for $j = 1, 2, \ldots, p$.

The power method is an iterative algorithm that starts with any real-valued vector $\mathbf{x}_0$ of length p and norm $||\mathbf{x}_0|| = 1$. The algorithm repeatedly computes products similar to those discussed in item 4 above, and the sequence of products converges to the first eigenvector.

To understand why the algorithm works, note that the eigenvectors of $\mathbf{A}$ are a basis for $\mathbb{R}^p$. Since $\mathbf{x}_0$ can be expressed as a linear combination of the eigenvectors of $\mathbf{A}$ (see equation 2.24) and a set of coefficients $\alpha_1, \ldots, \alpha_p$, the product of $\mathbf{A}$ and $\mathbf{x}_0$ can be expressed as

$$\begin{aligned}\mathbf{x}_1 &= \mathbf{A}\mathbf{x}_0 \\ &= \mathbf{A}\sum_i \alpha_i \mathbf{v}_i \\ &= \sum_i \alpha_i \mathbf{A}\mathbf{v}_i = \sum_i \alpha_i \lambda_i \mathbf{v}_i,\end{aligned}$$

The eigenvalue λ_1 can be factored out of the right hand sum:

$$\mathbf{x}_1 = \lambda_1 \sum_i \alpha_i \frac{\lambda_i}{\lambda_1} \mathbf{v}_i. \tag{2.26}$$

Another pre-multiplication by $\mathbf{A}$ and factorization yields

$$\mathbf{x}_2 = \mathbf{A}\mathbf{x}_1 = \lambda_1^2 \sum_i \alpha_i \left(\frac{\lambda_i}{\lambda_1}\right)^2 \mathbf{v}_i. \tag{2.27}$$

If we continue pre-multiplying and factoring, say k times in total, we arrive at

$$\mathbf{x}_{k+1} = \mathbf{A}^k \mathbf{x} = \lambda_1^k \sum_i \alpha_i \left(\frac{\lambda_i}{\lambda_1}\right)^k \mathbf{v}_i. \tag{2.28}$$

Assume that λ_1 is larger than all other eigenvalues. Then, for every $i > 1$, $(\lambda_i/\lambda_1)^k \approx 0$ for some sufficiently large value of k. Thus, for sufficiently large k,

$$\mathbf{x}_{k+1} = \mathbf{A}^k \mathbf{x}_k \approx \lambda_1^k \alpha_1 \mathbf{v}_1 \tag{2.29}$$

Ignoring machine error, $\mathbf{x}_{k+1} = \lambda_1^k \alpha_1 \mathbf{v}_1$ and $||\mathbf{x}_{k+1}|| = ||\lambda_1^k \alpha_1 \mathbf{v}_1||$. If we scale both vectors to have norm 1, we obtain

$$\frac{\mathbf{x}_{k+1}}{||\mathbf{x}_{k+1}||} = \frac{\lambda_1^k \alpha_1 \mathbf{v}_1}{||\mathbf{x}_{k+1}||} = \mathbf{v}_1. \tag{2.30}$$

Thus, sufficiently many repetitions of the product and factorization steps will yield

$$\mathbf{v}_1 = \mathbf{x}_{k+1}/||\mathbf{x}_{k+1}||. \tag{2.31}$$

The eigenvalue of $\mathbf{A}$ associated with $\mathbf{v}_1$ is (approximately) $\lambda_1 = \mathbf{v}_1^T\mathbf{A}\mathbf{v}_1$.

To implement this relationship, our algorithm begins with any non-zero vector $\mathbf{x}_0 \in \mathbb{R}$. The iterations are indexed by $k \in \{0, 1, \ldots\}$, and the updating step computes

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k/||\mathbf{x}_k||. \tag{2.32}$$

When the iterations are complete, we set $\mathbf{v}_1 = \mathbf{x}_{k+1}/||\mathbf{x}_{k+1}||$ and $\lambda_1 = \mathbf{v}_1^T\mathbf{A}\mathbf{v}_1$. We may end iterations when the norm of $\mathbf{x}_{k+1} - \mathbf{x}_k$ is small, say less than 10^{-7}.

If the second eigenvector pair is needed, the steps above are repeated using the residual matrix $\mathbf{R}_1 = \mathbf{A} - \lambda_1\mathbf{v}_1\mathbf{v}_1^T$. All eigenvector pairs can be computed in this fashion. If the matrix $\mathbf{A}$ is invertible, then the inverse may be computed as $\mathbf{A}^{-1} = \sum \lambda_i^{-1}\mathbf{v}_i\mathbf{v}_i^T$ (see exercise 6).

The following `Numpy` function will compute a chosen number of eigenvector pairs using the power method. The arguments passed to the function are the matrix (`S`), the number of iterations (100 ought to suffice), and the number of eigenvector pairs to extract (`nVectors`).

```
def powerMethod(S, nIter, nVectors):
    p = dim(S)[0]
    vectors = np.matrix(np.zeros(shape = (p, nVectors)))
    values = np.zeros(shape = (nVectors, 1))
    M = S.copy()

    for i in range(nVectors):
        x = np.matrix(np.ones(shape = (p, 1) ))*np.sqrt(1/p)
        for it in range(nIter):
            norm = np.linalg.norm(x)
            x = M*x/norm
        vectors[:,i] = x/np.linalg.norm(x)
        values[i] = vectors[:,i].T*S*vectors[:,i]
        M -= values[i][0] * vectors[:,i] * vectors[:,i].T
    pcAcct = np.cumsum(values)/sum(np.diag(S))
    printString = '  '.join([str(round(100*pct,1)) for pct in pcAcct])
    print('Cumulative % accounted for : '+printString)

    return values, vectors
```

2.9 Exercises

1. Build a symmetric $p \times p$ matrix for some value of $50 \leq p \leq 1000$ (a correlation or variance matrix will suffice). Use the power method and `np.linalg.eig` to compute the first few eigenvectors. What is the smallest number of iterations of the power method that produces a very good approximation of the correct (`np.linalg.eig`) eigenvectors? Compute and report the cosine between the eigenvectors computed by the two methods for, say $1, 2, 4, 8, \ldots, 128$ iterations. Your choice of number of iterations should show both poor and good approximations. Warning: if $\mathbf{v}$ is an eigenvector with eigenvalue λ, then $-\mathbf{v}$ is also an eigenvector with eigenvalue λ.
2. Show that orthogonal vectors are linearly independent. Hint: prove by contradiction.
3. Suppose that the column vectors of the $p \times p$ matrix $\mathbf{B} = [\mathbf{b}_1 \cdots \mathbf{b}_p]$ are a basis for $\mathbb{R}^p$ and $\mathbf{x} = x_1\mathbf{e}_1 + \cdots + x_p\mathbf{e}_p$. Give a formula for determining the coefficients $\alpha_1, \ldots, \alpha_p$ such that $\mathbf{x} = \sum \alpha_i \mathbf{b}_i$.
4. Suppose that $\underset{p\times 1}{\mathbf{v}} \neq \mathbf{0}$ where $p > 1$, and $\mathbf{A} = \mathbf{v}\mathbf{v}^T$. Show that $\mathbf{A}$ has has rank 1 by demonstrating that any column of $\mathbf{A}$ can be expressed as the product of the first column vector $\mathbf{v}_1$ and scalar. In other words, there is some β_j such that $\mathbf{v}_j = \beta_j \mathbf{v}_1$.
5. Prove that all eigenvalues of $\mathbf{A}$ are greater than zero if $\mathbf{A} = \mathbf{X}^T\mathbf{X}$ and $\mathbf{X}$ is full rank. Hint: use the original condition that $\mathbf{A} = \mathbf{X}^T\mathbf{X}$ and the full rank condition.
6. Prove that the inverse of $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$ is $\mathbf{V}\mathbf{\Lambda}^{-1}\mathbf{V}^T$.
7. Suppose that $(\lambda_i, \mathbf{v}_i)$ is an eigenvector pair of $\mathbf{A}$ and set $\mathbf{B} = \mathbf{A} - \lambda_i \mathbf{v}_i \mathbf{v}_i^T$.
   a. Prove that $(0, \mathbf{v}_i)$ is an eigenvector pair of $\mathbf{B}$.
   b. Suppose that $(\lambda_j, \mathbf{v}_j)$ is an eigenvector pair of $\mathbf{A}$ and $\mathbf{v}_j \neq \mathbf{v}_i$. Prove that $(\lambda_j, \mathbf{v}_j)$ is an eigenvector pair of $\mathbf{B}$.
8. This exercise demonstrates that the spectral decomposition of a symmetric matrix $\mathbf{A}$ can be written either as $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$ or as $\mathbf{A} = \sum_j \lambda_j \mathbf{v}_{\cdot j}\mathbf{v}_{\cdot j}^T$. Begin by computing the eigenvector pairs from a symmetric matrix $\mathbf{A}$ of your choice for which $p = 4$. With the eigenvectors $\mathbf{v}_{\cdot 1}, \mathbf{v}_{\cdot 2}, \ldots, \mathbf{v}_{\cdot p}$, form the matrix $\underset{p\times p}{\mathbf{V}} = [\mathbf{v}_{\cdot 1}\ \mathbf{v}_{\cdot 2}\ \cdots\ \mathbf{v}_{\cdot p}]$ and with the eigenvalues form the diagonal matrix

$$\mathbf{\Lambda} = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_p \end{bmatrix}.$$

   a. Confirm that

$$\mathbf{V\Lambda} = \begin{bmatrix} \lambda_1 v_{1,1} & \lambda_2 v_{2,1} & \cdots & \lambda_p v_{p,1} \\ \lambda_1 v_{1,2} & \lambda_2 v_{2,2} & \cdots & \lambda_p v_{p,2} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_1 v_{1,p} & \lambda_2 v_{2,p} & \cdots & \lambda_p v_{p,p} \end{bmatrix}$$

by computing $\mathbf{V\Lambda}$ and $\lambda_1 v_{1,1}, \lambda_2 v_{2,1}$ and $\lambda_1 v_{1,2}$.

b. Confirm that

$$\mathbf{V\Lambda V}^T = \begin{bmatrix} \sum_j \lambda_j v_{1,j}^2 & \sum_j \lambda_j v_{1,j} v_{2,j} & \cdots & \sum_j \lambda_j v_{1,j} v_{p,j} \\ \sum_j \lambda_j v_{2,j} v_{1,j} & \sum_j \lambda_j v_{2,j}^2 & \cdots & \sum_j \lambda_j v_{2,j} v_{p,j} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_j \lambda_j v_{p,j} v_{1,j} & \sum_j \lambda_j v_{p,j} v_{2,j} & \cdots & \sum_j \lambda_j v_{p,j}^2 \end{bmatrix}$$

by computing $\mathbf{V\Lambda V}^T$ and $\sum_j \lambda_j v_{1,j}^2$ and $\sum_j \lambda_j v_{2,j} v_{1,j}$.

c. Confirm that

$$\mathbf{V\Lambda V}^T = \sum_j \lambda_j \mathbf{v}_{\cdot j} \mathbf{v}_{\cdot j}^T,$$

by computing $\sum_j \lambda_j \mathbf{v}_{\cdot j} \mathbf{v}_{\cdot j}^T$.

Chapter 3
Cluster Analysis

Abstract Sometimes it's possible to divide a collection of observations into distinct subgroups based on nothing more than the observation attributes. If this can be done, then understanding the population or process generating the observations becomes easier. The intent of cluster analysis is to carry out a division of a data set into *clusters* of observations that are more alike within cluster than between clusters. Clusters are formed either by aggregating observations or dividing a single glob of observations into a collection of smaller sets. The process of cluster formation involves two varieties of algorithms. The first is measuring similarity between observations and between clusters. The second process is that of merging sets and it's dual, splitting sets. In this chapter, we discuss two popular cluster analysis algorithms: the k-means algorithm and hierarchical agglomerative clustering.

3.1 Introduction

Cluster analysis is a collection of methods aimed at the task of forming groups where none exist. For example, we may ask whether there are several distinct types of visitors to a website, say, visitors that browse but don't buy, customers that infrequently visit the website and buy only items from a particular department, and customers that make frequent visits to a website and buy a wide variety of items. If so, then the visitors of the first and second group might be offered incentives aimed at shifting them to the third group. Realistically though, this division may not exist in reality. It may be some other groups exist or that there is no clear divisions among visitors. With a set of observations on visitors and their behaviors, cluster analysis may provide insight into the structure of the visitor population and whether subgroups are manifested.

Hypothetically, the population of interest can be divided into meaningfully different groups, but usually, the analyst does know what the groups are,

and does not have representative observations from the groups. Therefore, the clusters must be formed without supervision. Clustering algorithms will form subsets by agglomerating observations that are most alike or splitting off groups that are most dissimilar based on the attributes measured on the variables. There's no force or function driving the cluster formation process towards meaningfully different groups.

Cluster analysis lacks a single, dominant mathematical objective. For comparison, linear regression is strongly driven by the linear model and the objective of minimizing the sum of squared differences between observations and fitted values. The mathematics and algorithms fall neatly into place from this starting point. In contrast, clustering algorithms have different approaches, all meritorious and occasionally useful. Even determining the appropriate number of clusters is difficult without prior knowledge of the population subdivisions. Despite these weaknesses, cluster analysis is still a useful tool of data analytics since the ability to examine groups of similar observations often sheds light on the population or process generating the data.

In this chapter, we sidestep the difficult issues related to the application and interpretation of cluster analysis and focus instead on the mechanics of two somewhat different but fundamental algorithms of cluster analysis: hierarchical agglomerative and k-means clustering. By understanding these basic algorithms, we learn of the strengths and weaknesses of cluster analysis.

Let us begin by defining $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ to be the data set.

3.1.1 Hierarchical Agglomerative Clustering

The hierarchical agglomerative approach begins with each observation defining a singleton cluster. Therefore, the initial set of clusters may be represented by $\{c_1, \ldots, c_n\}$ where $c_k = \{\mathbf{x}_k\}$ and n is the number of observations. The algorithm iteratively reduces the number of clusters by merging similar clusters. On the ith iteration, a cluster $c_{a\cup b}$ is formed by merging two clusters c_a and c_b, say,

$$(c_a, c_b) \longrightarrow c_a \cup c_b. \tag{3.1}$$

The choice of clusters to merge is determined by computing a distance between clusters and merging the pair that have the minimum inter-cluster distance. Consequently, a metric is needed to measure the distance between clusters. For example, the distance between the two clusters may be smallest distance between any two vector belonging c_a and any vector belonging to c_b. Mathematically, this distance is defined as

$$d_1(c_a, c_b) = \min_{\mathbf{x}_k \in c_a, \mathbf{x}_l \in c_b} d_C(\mathbf{x}_k, \mathbf{x}_l). \tag{3.2}$$

where $d_C(\mathbf{x}, \mathbf{y})$ is the city-block distance between vectors $\mathbf{x}$ and $\mathbf{y}$ defined by equation (**??**). There's nothing special about city-block distance—other metrics possible and perhaps preferable. The minimum distance metric d_1 tends to produce string-like clusters. More compact clusters result from a centroid-based metric that utilizes the cluster centroids

$$\begin{aligned} \overline{\mathbf{x}}_a &= n_a^{-1} \sum_{\mathbf{x}_k \in c_a} \mathbf{x}_k \\ \overline{\mathbf{x}}_b &= n_b^{-1} \sum_{\mathbf{x}_k \in c_b} \mathbf{x}_k, \end{aligned} \tag{3.3}$$

where n_a and n_b are the numbers of observations in c_a and c_b, respectively. Then, the distance between c_a and c_b is

$$d_{\text{ave}}(c_a, c_b) = d_C(\overline{\mathbf{x}}_a, \overline{\mathbf{x}}_b). \tag{3.4}$$

As described above, the algorithm will progressively merge clusters until there is a single cluster. Merging clusters into a single glob is only interesting if the sequence of merges is interesting. If the clusters are to be used for some purpose, then it's necessary to terminate the algorithm while there are still some clusters remaining.

3.1.2 The k-Means Algorithm

Our example of representative-based cluster analysis is the k-means algorithm. The algorithm begins with setting the number of clusters to be k and randomly selecting k vectors, each serving as a representative, or *centriod* of one cluster. The initial set of clusters may be represented by $\{c_1, \ldots, c_k\}$ where $c_i = \{\mathbf{x}_i\}$. The Euclidean distance between $\mathbf{x}_j \in D$ and c_a is defined to be the distance between $\mathbf{x}_j$ and the centroid $\mathbf{x}_a$ of c_a:

$$d_E(\mathbf{x}_j, c_a) = \left[(\mathbf{x}_j - \mathbf{x}_a)^T (\mathbf{x}_j - \mathbf{x}_a)\right]^{1/2}. \tag{3.5}$$

Every observation is assigned to the cluster nearest to it according to the Euclidean distances.[1] Upon completion of the initialization phase, the algorithm begins iterating between two steps:

The first step updates the cluster centroid as the vector mean of every member:

$$\overline{\mathbf{x}}_a = n_a^{-1} \sum_{\mathbf{x}_k \in c_a} \mathbf{x}_k, \tag{3.6}$$

where n_a is the number of observations belonging to c_a. The second step iterates over D and computes the Euclidean distances between each $\mathbf{x}_j \in D$ and each cluster centroid. If an observation is found to be nearest to a different

[1] We may use the squared Euclidean distance instead since the ordering will be the same.

cluster than its currently assigned cluster, then we reassign it to the nearest cluster. If any observation is reassigned, then another iteration commences. The algorithm terminates when no further changes in membership occur.

3.2 Exercises

1. One aspect of cluster analysis that was not addressed in detail in class is the assessment of a particular clustering algorithm. While the objective function is a helpful statistic, a measure of intra-cluster variability *for each cluster* provides more information.
   One example of a measure of intra-cluster variability is the mean Euclidean distance between members of a cluster (averaged over all pairs that may be formed from the cluster members). The mathematical definition is
   $$f_1(C) = \binom{n}{2}^{-1} \sum_{\mathbf{x}_i, \mathbf{x}_j \in C} d_E(\mathbf{x}_i, \mathbf{x}_j), \tag{3.7}$$
   where n is the number of members belonging to cluster C and $\binom{n}{2}$ is the number of unique sets that may be constructed when choosing 2 elements from n. Note that if there are k clusters, then one may compute $f_1(C_1), f_1(C_2), \ldots, f_1(C_k)$.
   Another measure, call it f_2, can be defined by replacing $d_E(\mathbf{x}_i, \mathbf{x}_j)$ with the cosine distance between $\mathbf{x}_i$ and $\mathbf{x}_j$ (defined in equation 3.9 below).
   A third measure of intra-cluster variability is the mean distance between each member and the cluster centroid $\overline{\mathbf{x}}$:
   $$f_3(C) = n^{-1} \sum_{\mathbf{x}_i \in C} d_E(\mathbf{x}_i, \overline{\mathbf{x}}). \tag{3.8}$$
   The objective of this exercise is to evaluate intra-cluster variability by computing and summarizing f_1, f_2, or f_3.
   Write `Python` code (preferably as a `Python` function) that computes one of f_1, f_2, or f_3. Modify `clusterAnaysis.py` so that your function is called at the completion of the k-means algorithm. Paste the code into your homework document and identify it.
2. Carry out clustering for some configuration of number of clusters and number of stocks of your choice. Compute the sum of the intra-cluster variability measure over all clusters as a single measure of clustering effectiveness, say $\sum_C f_k(C)$, where k is the number of cluster. Make a table tabulating the cluster label $(1, 2, \ldots, k)$, the number of members in the cluster, and intra-cluster variability for the cluster. Report the number of stocks and the series length (number of observations for each stock).

3. Apply the intra-cluster variability measure when forming 3, 4, and 5 clusters[2]. Use the same starting configuration for each execution of the k-means algorithm (except for number of clusters). Report the sum of the intra-cluster variability measures for 3, 4, and 5 clusters.
4. Theoretically, how should the mean distances from each member to the cluster centroid vary with number of clusters? If the aim of cluster analysis is to find a set of stocks each of which is to be used to forecast itself (that is, to predict future values), and the other stocks in the set, then is small intra-cluster variability desirable? Are there (potentially) any undesirable aspects to forcing intra-cluster variability to be small? Explain your reasoning.
5. Change the objective function in `clusterAnaysis.py` so that there is an option to minimize the sum of the cosine distances between the series of stock values and the centroids. Cosine distance can be computed as

$$d(\mathbf{x}, \overline{\mathbf{x}}) = \frac{1 - r(\mathbf{x}, \overline{\mathbf{x}})}{2}, \tag{3.9}$$

   where $r(\mathbf{x}, \overline{\mathbf{x}})$ is the Pearson's correlation between the vector $\mathbf{x}$ and the cluster mean vector $\overline{\mathbf{x}}$. The `Numpy` function `corrcoef` will compute the correlation between two vectors (or the rows of a matrix), thereby making the calculation of $d(\mathbf{x}, \overline{\mathbf{x}})$ very easy to implement.
   Two changes must be made: the cosine distance function must be used to determine observation to centroid distance and the objective function must be changed from Euclidean distance between the sum of the cluster centroid-to-member distances to the sum of all possible cosine distances. Using cosine distance, form 3, 4, and then 5 clusters. Use the same starting configuration for each execution of the k-means algorithm. For one choice of number of clusters, provide two plots showing the k cluster centroids plotted against day (one for Euclidean distance and the second for cosine distance).
6. Change the objective function in `clusterAnaysis.py` so that the k-means algorithm minimizes the sum of the Hamming distances between the series of inter-day price *changes* and the cluster centroid.
   First, replace the standardized prices with the binary outcomes

$$b(y_i) = \begin{cases} 1, \ \texttt{if} \ x_{i+1} > x_i \\ 0, \ \texttt{if} \ x_{i+1} \leq x_i \end{cases}, i = 1, 2, \ldots, n-1. \tag{3.10}$$

   Note that there will be one fewer binary outcomes than standardized prices for a particular stock. It's probably best to replace the elements in the matrix $\mathbf{X}$ immediately after it is computed in the `getData` function since the observations are arranged in chronological order. Do not

[2] You may set some other number of clusters if you like.

compute Z. The function ought to return the matrix of binary outcomes instead of Z.

Compute the cluster centroid as a series of Hamming distances means (averaged across the cluster members). Assess the results of the change by by producing two plots showing the cluster centroids plotted against day, one generated using for Hamming distance and the second generated using the usual Euclidean (or cosine) distance.

7. Change the distance function from Euclidean distance to a weighted Euclidean distance that places the most weight on the most recent observations. Use an exponential weighting function

$$w_i = \alpha(1-\alpha)^{n-i}, i = 1, 2, \ldots, n, \tag{3.11}$$

where nth observation is the most recent observation and $0 < \alpha < 1$. A choice of α between .05 and .2 seems reasonable.

   a. Create plots of the cluster centroids after initialization of the centroids (and before) executing the k-means algorithm and then again after completion of the algorithm. Note: unless the objective function is modified to reflect the new distance, don't expect the objective function to decrease monotonically with iteration.
   b. Verify computationally that $\sum_{i=1}^{n} w_i \approx 1$. In other words, compute the sum of the w_i's. Provide the code that computes the sum and report the sum.
   c. (Math students) Verify mathematically that $\sum_{i=0}^{\infty} \alpha(1-\alpha)^i = 1$.

8. Consider two vectors $\mathbf{x}$ and $\mathbf{y} = c\mathbf{x}$, for $c \neq 0$. Show that the cosine of the angle between the two vectors is 1, and hence, the cosine distance between the vectors is 0.
9. Assume that $\mathbf{x}_i$, $\mathbf{y}$, and $\overline{\mathbf{x}}_i$ are $m \times 1$ vectors and that $\overline{\mathbf{x}}_i$ is the vector mean of $\mathbf{x}_1, \ldots, \mathbf{x}_n$. Show that

$$\sum_{i=1}^{n} ||\mathbf{x}_i - \mathbf{y}||^2 = \sum_{i=1}^{n} ||\mathbf{x}_i - \overline{\mathbf{x}}||^2 + n||\mathbf{y} - \overline{\mathbf{x}}||^2, \tag{3.12}$$

where $||\mathbf{x}|| = (\mathbf{x}^T\mathbf{x})^{1/2}$ is the L_2 norm of the $m \times 1$ vector $\mathbf{x}$.

Chapter 4
Finite Mixtures

4.1 Introduction

We introduce the finite mixtures approach to unsupervised classification. The following discussion assumes that the observed data consist of observations belonging to one of C classes though the classes are unknown as there are no observations on membership. The objective is to elucidate the classes by determining the estimated probability of membership in the unknown groups for each observation. In this chapter, we concentrate on observational data consisting of on a one-hot mulitnomial random variable that identifies membership in one of c classes. Therefore, the observational or experimental units generating the observations and the observation vector identifies membership via the one-hot observation.

4.2 Finite Mixtures

Consider a vector random variable $\mathbf{x} = [x_1 \ \cdots \ x_c]$ consisting of c independent Bernoulli random variables and suppose that $\mathbf{x}$ may be generated or drawn from one of K distributions or subpopulations. We allow $\Pr(x_j = 1)$ to depend on the generating distribution or sampled subpopulation. Therefore, a second subscript (k) is introduced to identify the distribution that generated the realization of $\mathbf{x}$, and write $\Pr(x_j = 1) = \pi_{j,k}$ provided that $\mathbf{x}$ was generated from the kth distribution. In summary, $\mathbf{x}$ is generated (or sampled) from one of K distributions, with possibly different Bernoulli probabilities $\pi_{j,k}, j = 1, 2, \ldots, c, k = 1, 2, \ldots, K$.

The random variable $\mathbf{z} = [z_1 \ \cdots z_k \ \cdots z_K]$ is a multinomial random variable that identifies the distribution that generated $\mathbf{x}$. Therefore, a realization of $\mathbf{z}$ is a one-hot vector and one of $\iota_1, \ldots, \iota_K$, where ι_k is a K-length vector of zeros except that position k is occupied by a 1. In other words, $\mathbf{z} = \iota_k$ if

and only if $z_k = 1$.[1] We introduce the parameters $\alpha_1, \ldots, \alpha_K$ and express the distribution of $\mathbf{z}$ as

$$\Pr(\mathbf{z} = \iota_k) = \prod_{k=1}^{c} \alpha_k^{z_k}, \tag{4.2}$$

where $0 < \alpha_k < 1, k = 1, \ldots, K$ and $\sum_k \alpha_k = 1$. It follows that the sample space of $\mathbf{z}$ is

$$\{\iota_1, \iota_2, \ldots, \iota_K\} = \left\{ \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \cdots \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \right\}. \tag{4.3}$$

Because $x_1, x_2, \ldots, x_C$ are assumed to be independent, the conditional distribution of $\mathbf{x}$ is

$$\begin{aligned} \Pr(\mathbf{x}|\mathbf{z}) &= \prod_{j=1}^{c} \Pr(x_j|\mathbf{z}) \\ &= \prod_{j=1}^{c} \left[\pi_{j,k}^{x_j} (1 - \pi_{j,k})^{(1-x_j)} \right]^{z_k}, \end{aligned} \tag{4.4}$$

and the joint distribution of $\mathbf{x}$ and $\mathbf{z}$ is

$$\begin{aligned} \Pr(\mathbf{x}, \mathbf{z}) &= \Pr(\mathbf{x}|\mathbf{z}) \Pr(\mathbf{z}) \\ &= \prod_{k=1}^{K} \Big(\prod_{j=1}^{c} \pi_{k,j}^{x_j} (1 - \pi_{k,j})^{(1-x_j)} \Big)^{z_k} \times \prod_{k=1}^{K} \alpha_k^{z_k} \\ &= \prod_{k=1}^{K} \Big(\alpha_k \prod_{j=1}^{c} \pi_{k,j}^{x_j} (1 - \pi_{k,j})^{(1-x_j)} \Big)^{z_k}. \end{aligned} \tag{4.5}$$

We'll need the conditional distribution of $\mathbf{z}$ given $\mathbf{x}$ and the conditional expectation of $\mathbf{z}$ given $\mathbf{x}$, so let us continue the calculations:

---

[1] Hence,

$$z_k = \begin{cases} 1 & \text{if } \mathbf{x} \text{ originates from the } k\text{th subpopulation,} \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

.

$$\begin{aligned}\Pr(\mathbf{z}=\iota_k|\mathbf{x}) =& \frac{\Pr(\mathbf{x},\mathbf{z}=\iota_k)}{\Pr(\mathbf{x})}\\ =& \frac{\Pr(\mathbf{z}=\iota_k)\Pr(\mathbf{x}|\mathbf{z}=\iota_k)}{\sum_l \Pr(\mathbf{z}=\iota_l)\Pr(\mathbf{x}|\mathbf{z}=\iota_l)}\\ =& \frac{\alpha_k \prod_j^c \pi_{k,j}^{x_j}}{\sum_l \alpha_l \prod_j^c \pi_{k,l}^{x_{i,j}}},\end{aligned} \tag{4.6}$$

and since a realization of z_k is either 0 or 1,

$$\begin{aligned}\mathrm{E}(z_k|\mathbf{x}) =& \sum_{z\in\{0,1\}} z\Pr(\mathbf{z}=\iota_k|\mathbf{x})\\ =& \Pr(\mathbf{z}=\iota_k|\mathbf{x}).\end{aligned} \tag{4.7}$$

The expected value of the vector random variable $\mathbf{z}$ consists of the terms shown in equations (4.6) and (4.7); hence,

$$\begin{aligned}\mathrm{E}(\mathbf{z}|\mathbf{x}) =& [\mathrm{E}(z_1|\mathbf{x}) \ \cdots \ \mathrm{E}(z_K|\mathbf{x})]\\ =& [\Pr(\mathbf{z}=\iota_1|\mathbf{x}) \ \cdots \ \Pr(\mathbf{z}=\iota_K|\mathbf{x})].\end{aligned} \tag{4.8}$$

It's important to note that the membership variable $\mathbf{z}$ is unobserved, and hence, a *latent* variable. At this juncture, the model is a theoretical construct since there's no obvious means of accounting for the latent variables in practice. It's application presents a significant challenge that is met by application of the EM algorithm. The main idea underlying the EM algorithm is that if estimates of the parameters constituting the set $\Omega = \{\alpha_1, \ldots, \alpha_K, \pi_{1,1} \ldots, \pi_{K,c}\}$ are available, then the conditional expectation $\mathrm{E}(\mathbf{z}|\mathbf{x})$ can be estimated. Further, the estimated conditional expectations can be used whenever an actual value of $\mathbf{z}$ is needed—in particular, to compute the maximum likelihood estimate of Ω. The EM algorithm iterates between computing the conditional expectations (the E-step) and computing the maximum likelihood estimators (the M-step).

Let's turn now to applying the probability model for the purpose of unsupervised classification. We'll suppose, without justification, that the data consist of N independent realizations of the vector random variable $\mathbf{x}$. Two data sets are identified: $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ consisting of the actual observations, and the *complete data* $D_c = \{(\mathbf{x}_1, \mathbf{z}_1), \ldots, (\mathbf{x}_N, \mathbf{z}_N)\}$'s. The complete data set is not observable the elements consist of the actual observations paired with the latent variables $\mathbf{z}_1, \ldots, \mathbf{z}_N$.

4.3 The E-step

In anticipation of determining the maximum likelihood estimators (despite $\mathbf{z}_1, \ldots, \mathbf{z}_N$ being unobserved), equation (4.5) leads to the natural logarithm of the joint distribution:

$$\begin{aligned}\log \Pr(\mathbf{x}_i, \mathbf{z}_i) = \sum_{k=1}^{K} z_{i,k} \big[\log(\alpha_k) \\ + \sum_{j=1}^{c} x_{i,j} \log(\pi_{k,j}) + (1 - x_{i,j}) \log(1 - \pi_{k,j}) \big] \big].\end{aligned} \tag{4.9}$$

The *complete-data* log-likelihood is

$$\begin{aligned}L(\Omega | D_c) &= \sum_{i}^{N} \log \Pr(\mathbf{x}_i, \mathbf{z}_i) \\ &= \sum_{k=1}^{K} z_{i,k} \big[\log(\alpha_k) \sum_{j=1}^{c} x_{i,j} \log(\pi_{k,j}) + (1 - x_{i,j}) \log(1 - \pi_{k,j}) \big].\end{aligned} \tag{4.10}$$

Since $\mathbf{z}_1, \ldots, \mathbf{z}_N$ are unobserved, it's not practical to determine the maximum likelihood estimators of Ω from $L(\Omega)$. However, if it were possible to arrive at the maximum likelihood estimators by some means, say using the observed data, then we may estimate the conditional expectations of $\mathbf{z}_1, \ldots, \mathbf{z}_N$. It's logical to predict $\mathbf{x}_i$ to belong to the group with the largest estimated conditional expectation, or estimated probability of group membership. In other words, if the maximum likelihood estimator of Ω were available, say, $\widehat{\Omega} = \{\widehat{\alpha}_1, \ldots, \widehat{\alpha}_K, \widehat{\pi}_{1,1}, \ldots, \widehat{\pi}_{K,c}\}$, then the *soft* classification of $\mathbf{x}_i$ might be

$$\arg\max\{\widehat{\Pr}(z_{i,1}|\mathbf{x}_i) \ldots, \widehat{\Pr}(z_{i,K}|\mathbf{x}_i)\}, \tag{4.11}$$

where

$$\widehat{\Pr}(z_{i,k}|\mathbf{x}_i) = \widehat{\mathrm{E}}(z_{i,k}|\mathbf{x}_i) = \frac{\widehat{\alpha}_k \prod_j^c \widehat{\pi}_{k,j}^{x_{i,j}}}{\sum_l \widehat{\alpha}_l \prod_j^c \widehat{\pi}_{l,j}^{x_{i,j}}}. \tag{4.12}$$

It's necessary to eliminate the latent variables from the complete-data maximum likelihood. (This step is necessary because the latent variables are unobserved and cannot be used for calculating the maximum likelihood estimators). The EM algorithm transformation replaces the complete-data log-likelihood by its conditional expectation given the observed data, that is $\mathrm{E}(L(\Omega|\mathbf{x}_1, \ldots, \mathbf{x}_N)$. By independence, the conditional expectation of the complete-data log-likelihood is a sum over the conditional expectations of the components:

$$\begin{aligned}\mathrm{E}(L(\Omega|\mathbf{x}_1,\ldots,\mathbf{x}_N) &= \sum_{i}^{N} \mathrm{E}[\log \Pr(\mathbf{x}_i,\mathbf{z}_i)|\mathbf{x}_i] \\ &= \sum_{i}^{N}\sum_{k}^{K} \mathrm{E}(z_{i,k}|\mathbf{x}_i)\Big[\log(\alpha_k)\sum_{j=1}^{c} x_{i,j}\log(\pi_{k,j}) \\ &\quad + (1-x_{i,j})\log(1-\pi_{k,j})\Big],\end{aligned} \tag{4.13}$$

where equation (4.6) provides a formula for calculating $\mathrm{E}(z_{i,k}|\mathbf{x}_i)$ that does not depend on the latent variables.

4.4 The M-step

The EM algorithm uses $\mathrm{E}(L(\Omega|\mathbf{x}_1,\ldots,\mathbf{x}_N)$ in place of $L(\Omega)$ to determine estimators of Ω. We proceed in the usual fashion of differentiating the objective function with respect to the parameters, setting the resulting gradient vector to $\mathbf{0}$ and solving for the parameters. Beginning $\alpha_1,\ldots,\alpha_k$, note that this set of parameters is constrained to sum to 1, and so to determine the maximizing value of $\mathrm{E}(L(\Omega|\mathbf{x}_1,\ldots,\mathbf{x}_N)$ given the constraint, we introduce Langrange multipliers to the objective function and set

$$\begin{aligned}g(\Omega,\lambda) = &\sum_{k=1}^{K} \mathrm{E}(z_{i,k}|\mathbf{x}_i)\Big[\log(\alpha_k)\sum_{j=1}^{c} x_{i,j}\log(\pi_{k,j}) \\ &+ (1-x_{i,j})\log(1-\pi_{k,j})\Big] - \lambda\Big(\sum_{k=1}^{K}\alpha_k - 1\Big).\end{aligned} \tag{4.14}$$

Then,

$$\frac{\mathrm{d}g(\Omega,\lambda)}{\mathrm{d}\alpha_k} = \sum_{i}^{N} \mathrm{E}(z_{i,k}|\mathbf{x}_i)\alpha_k^{-1} - \lambda. \tag{4.15}$$

Setting $\mathrm{d}g(\Omega,\lambda)/\mathrm{d}\alpha_k = 0$ and rearranging yields, Then, for each $k = 1,\ldots,K$

$$\begin{aligned}\lambda\alpha_k &= \sum_{i}^{N} \mathrm{E}(z_{i,k}|\mathbf{x}_i) \\ \Rightarrow \lambda\sum_{k}\alpha_k &= \sum_{i}^{N}\sum_{k} \mathrm{E}(z_{i,k}|\mathbf{x}_i)\end{aligned} \tag{4.16}$$

Note that $\sum_k \alpha_k = 1$ and that $\mathrm{E}(z_{i,k}|\mathbf{x}_i) = \Pr(z_{i,k} = 1|\mathbf{x}_i)$. Therefore, $\sum_k \mathrm{E}(z_{i,k}|\mathbf{x}_i) = 1$, for each $k = 1,\ldots,N$. Hence, $\lambda = N$ and the solution to equation (4.15) is

$$\widehat{\alpha}_k = N^{-1} \sum_i^N \mathrm{E}(z_{i,k}|\mathbf{x}_i). \tag{4.17}$$

Let's next determine estimators for $\pi_{k,j}$. Differentiating the complete-data log-likelihood with respect to $\pi_{k,j}$ yields

$$\begin{aligned} \frac{\mathrm{d}L(\Omega,\lambda)}{\mathrm{d}\pi_{k,j}} &= \sum_i^N \mathrm{E}(z_{i,k}|\mathbf{x}_i) \log(\alpha_k) \left(\frac{x_{i,j}}{\pi_{k,j}} - \frac{1-x_{i,j}}{1-\pi_{k,j}} \right) \\ &= \sum_i^N \mathrm{E}(z_{i,k}|\mathbf{x}_i) \left(\frac{x_{i,j}}{\pi_{k,j}} - \frac{1-x_{i,j}}{1-\pi_{k,j}} \right) \end{aligned} \tag{4.18}$$

Setting $\mathrm{d}L(\Omega,\lambda)/\mathrm{d}\pi_{k,j} = 0$ implies

$$\frac{\sum_i \mathrm{E}(z_{i,k}|\mathbf{x}_i) x_{i,j}}{\pi_{k,j}} = \frac{\mathrm{E}(z_{i,k}|\mathbf{x}_i)[1 - \sum_i x_{i,j}]}{1-\pi_{k,j}}. \tag{4.19}$$

Solving equation (4.19) for $\pi_{k,j}$ yields the solution

$$\widehat{\pi}_{k,j} = \frac{\sum_i \mathrm{E}(z_{i,k}|\mathbf{x}_i) x_{i,j}}{\sum_i \mathrm{E}(z_{i,k}|\mathbf{x}_i)}. \tag{4.20}$$

The finite mixture EM algorithm begins by setting initial values for $\mathrm{E}(z_{i,k}|\mathbf{x}_i) = \Pr(z_{i,k}|\mathbf{x}_i)$, for each i and k. One may choose random numbers from the uniform distribution on (0, 1) and then scale the values so that $\sum_{k=1}^K \widehat{\mathrm{E}}(z_{i,k}|\mathbf{x}_i) = 1$ for each $i = 1, \ldots, N$. Then, the algorithm iterates between the E- and M-steps. To be more precise, lets suppose that the iterations are indexed by κ and that the initial estimates of the expectations are denoted as $\mathrm{E}^{(0)}(z_{i,k}|\mathbf{x}_i)$. Then, the initial parameter estimates are computed according to

$$\alpha_k^{(0)} = N^{-1} \sum_i^N \mathrm{E}^{(0)}(z_{i,k}|\mathbf{x}_i). \tag{4.21}$$

and

$$\widehat{\pi}_{k,j}^{(0)} = \frac{\sum_i \mathrm{E}^{(0)}(z_{i,k}|\mathbf{x}_i) x_{i,j}}{\sum_i \mathrm{E}^{(0)}(z_{i,k}|\mathbf{x}_i)}. \tag{4.22}$$

Each iteration, $\kappa = 1, 2, \ldots$, computes, in order,

$$\begin{aligned}
\mathrm{E}^{(\kappa)}(z_{i,k}|\mathbf{x}_i) =& \frac{\alpha_k^{(\kappa-1)} \prod_j^c \pi_{k,j}^{(\kappa-1)^{x_{i,j}}}}{\sum_l \widehat{\alpha}_l \prod_j^c \pi_{l,j}^{(\kappa-1)^{x_{i,j}}}}, \\
\alpha_k^{(\kappa)} =& N^{-1} \sum_i^N \mathrm{E}^{(\kappa)}(z_{i,k}|\mathbf{x}_i), \\
\widehat{\pi}_{k,j}^{(\kappa)} =& \frac{\sum_i \mathrm{E}^{(\kappa)}(z_{i,k}|\mathbf{x}_i) x_{i,j}}{\sum_i \mathrm{E}^{(\kappa)}(z_{i,k}|\mathbf{x}_i)}.
\end{aligned} \tag{4.23}$$

4.5 Exercises

1. Suppose that $\{x_1, \ldots, x_n\}$ are independent random variables with a common domain (for simplicity, you may choose the random variables to be all continuous or all discrete). Suppose that f is a function defined on the domain. Show that the conditional expectation of the sum is equal to the sum of the conditional expectations:

$$\mathrm{E}\Big[\sum f(x_i)|x_1, \ldots, x_n\Big] = \sum \mathrm{E}[f(x_i)|x_i], \tag{4.24}$$

Chapter 5
Matrix Differentiation

Suppose that $\mathbf{x}$ is a q-length vector and $\mathbf{y} = f(\mathbf{x})$ is a p-length vector. The *Jacobian* matrix of the map $f : \mathbb{R}^q \to \mathbb{R}^p$ is the matrix of partial derivatives

$$\underset{p\times q}{\frac{\partial \mathbf{y}}{\partial \mathbf{x}^T}} = \begin{bmatrix} \frac{\mathrm{d}y_1}{\mathrm{d}x_1} & \frac{\mathrm{d}y_1}{\mathrm{d}x_2} & \cdots & \frac{\mathrm{d}y_1}{\mathrm{d}x_q} \\ \frac{\mathrm{d}y_2}{\mathrm{d}x_1} & \frac{\mathrm{d}y_2}{\mathrm{d}x_2} & \cdots & \frac{\mathrm{d}y_2}{\mathrm{d}x_q} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\mathrm{d}y_p}{\mathrm{d}x_1} & \frac{\mathrm{d}y_p}{\mathrm{d}x_2} & \cdots & \frac{\mathrm{d}y_p}{\mathrm{d}x_q} \end{bmatrix}. \tag{5.1}$$

It's convenient to write the Jacobian as a matrix of columns

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}^T} = \begin{bmatrix} \underset{p\times 1}{\frac{\partial \mathbf{y}}{\partial x_1}} & \underset{p\times 1}{\frac{\partial \mathbf{y}}{\partial x_2}} & \cdots & \underset{p\times 1}{\frac{\partial \mathbf{y}}{\partial x_q}} \end{bmatrix} \tag{5.2}$$

or a matrix of rows

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}^T} = \begin{bmatrix} \underset{1\times q}{\frac{\partial y_1}{\partial \mathbf{x}^T}} \\ \vdots \\ \underset{1\times q}{\frac{\partial y_p}{\partial \mathbf{x}^T}} \end{bmatrix} \tag{5.3}$$

depending on the situation.

The first and simplest example is

$$\frac{\partial \mathbf{x}}{\partial \mathbf{x}^T} = \mathbf{I}_p.$$

5.1 Some useful results

1. Suppose that $\mathbf{A}$ is a $p \times q$ matrix of real numbers and $\mathbf{y} = \mathbf{A}\mathbf{x}$. Then $y_i = \mathbf{a}_{i\cdot}\mathbf{x}$, where $\mathbf{a}_{i\cdot}$ is the ith row of $\mathbf{A}$. If we use the fact that $y_i = \mathbf{a}_{i\cdot}\mathbf{x}$ and
$$\begin{aligned}\frac{\partial x_j}{\partial \mathbf{x}^T} &= \underset{1\times q}{\left[\frac{\mathtt{d}x_j}{\mathtt{d}x_1} \ \cdots \frac{\mathtt{d}x_j}{\mathtt{d}x_j} \ \cdots \ \frac{\mathtt{d}x_j}{\mathtt{d}x_q}\right]} \\ &= [0 \ \cdots \ 1 \ \cdots \ 0],\end{aligned} \tag{5.4}$$
where the 1 appears in column j, then it follows that
$$\begin{aligned}\frac{\partial y_i}{\partial \mathbf{x}^T} &= \frac{\partial \sum_j^q a_{i,j}x_j}{\partial \mathbf{x}^T} \\ &= \sum_{j=1}^{q} a_{i,j} \underset{1\times q}{\frac{\partial x_j}{\partial \mathbf{x}^T}} \\ &= [a_{i,1} \ a_{i,2} \ \cdots \ a_{i,q}] = \mathbf{a}_{i\cdot}.\end{aligned} \tag{5.5}$$
Thus,
$$\frac{\partial \mathbf{A}\mathbf{x}}{\partial \mathbf{x}^T} = \begin{bmatrix} \frac{\partial \mathbf{a}_{1\cdot}\mathbf{x}}{\partial \mathbf{x}^T} \\ \vdots \\ \frac{\partial \mathbf{a}_{p\cdot}\mathbf{x}}{\partial \mathbf{x}^T} \end{bmatrix} = \begin{bmatrix} \mathbf{a}_{1\cdot} \\ \mathbf{a}_{2\cdot} \\ \vdots \\ \mathbf{a}_{p\cdot} \end{bmatrix} = \mathbf{A}. \tag{5.6}$$

2. Consider two multivariate functions $\mathbf{z}(\cdot)$ and $\mathbf{y}(\cdot)$ and suppose that $f(\mathbf{x}) = \mathbf{z}(\mathbf{x})^T\mathbf{y}(\mathbf{x}) = \mathbf{y}(\mathbf{x})^T\mathbf{z}(\mathbf{x})$. Let us determine the Jacobian of $f(\mathbf{x})$. First, note that
$$f(\mathbf{x}) = \sum_{i=1}^{p} z_i(\mathbf{x})y_i(\mathbf{x}). \tag{5.7}$$
Using the product rule, the derivative of $f(\mathbf{x})$ with respect to x_i is
$$\frac{\mathtt{d}f(\mathbf{x})}{\mathtt{d}x_i} = \frac{\mathtt{d}z_i(\mathbf{x})}{\mathtt{d}x_i}y_i(\mathbf{x}) + \frac{\mathtt{d}y_i(\mathbf{x})}{\mathtt{d}x_i}z_i(\mathbf{x}). \tag{5.8}$$
Hence, the vector of partial derivative can be expressed as
$$\underset{p\times 1}{\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}} = \underset{p\times p}{\frac{\partial \mathbf{z}(\mathbf{x})}{\partial \mathbf{x}}}^T \underset{p\times 1}{\mathbf{y}(\mathbf{x})} + \underset{p\times p}{\frac{\partial \mathbf{y}(\mathbf{x})}{\partial \mathbf{x}}}^T \underset{p\times 1}{\mathbf{z}(\mathbf{x})}. \tag{5.9}$$
Now consider the quadratic form $f(\mathbf{x}) = \mathbf{x}^T\mathbf{A}\mathbf{x}$ and it's Jacobian with respect to $\mathbf{x}$. We'll apply equation (5.9) by setting

$$\begin{aligned} \mathbf{z}(\mathbf{x}) &= \mathbf{x} \\ \mathbf{y}(\mathbf{x}) &= \mathbf{A}\mathbf{x}. \end{aligned} \tag{5.10}$$

Thus, $f(\mathbf{x}) = \mathbf{z}(\mathbf{x})^T\mathbf{y}(\mathbf{x})$, and

$$\begin{aligned} \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} &= \frac{\partial \mathbf{z}(\mathbf{x})}{\partial \mathbf{x}}^T \mathbf{y}(\mathbf{x}) + \frac{\partial \mathbf{y}(\mathbf{x})}{\partial \mathbf{x}}^T \mathbf{z}(\mathbf{x}) \\ &= \mathbf{I}_p\mathbf{A}\mathbf{x} + \mathbf{A}^T\mathbf{x} \\ &= \left(\mathbf{A} + \mathbf{A}^T\right)\mathbf{x}. \end{aligned} \tag{5.11}$$

If $\mathbf{A}$ is symmetric, then

$$\frac{\partial \mathbf{x}^T\mathbf{A}\mathbf{x}}{\partial \mathbf{x}} = 2\mathbf{A}\mathbf{x}. \tag{5.12}$$

3. Consider a $p \times q$ matrix that depends on x, say $\mathbf{A}(x)$. We define the Jacobian of $\mathbf{A}$ with respect to x to be the matrix

$$\underset{p\times q}{\frac{\partial \mathbf{A}(x)}{\partial x}} = \left[\underset{p\times q}{\frac{\mathtt{d}a_{i,j}(x)}{\mathtt{d}(x)}}\right]. \tag{5.13}$$

5.2 Example

The least squares objective function for a p-length parameter vector $\boldsymbol{\beta}$ is

$$\varepsilon(\boldsymbol{\beta}) = \sum_{i=1}^{n}(y_i - \mathbf{x}_i^T\boldsymbol{\beta})^2, \tag{5.14}$$

where $\mathbf{x}_i$ is a p-vector of predictors and y_i is the associated observation on the target variable. In matrix form, the objective function is

$$\varepsilon(\boldsymbol{\beta}) = (\mathbf{y} - \mathbf{X}^T\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}^T\boldsymbol{\beta}), \tag{5.15}$$

where $\mathbf{y} = [y_1\ y_2\ \cdots\ y_n]^T$ is n-vector and $\mathbf{X}$ is $n \times p$ and constructed by stacking the n predictor vectors as rows. The least squares estimator of $\boldsymbol{\beta}$ is determined by differentiating $\varepsilon(\boldsymbol{\beta})$ with respect to $\boldsymbol{\beta}$, setting the vector of derivatives equal to $\mathbf{0}$, and solving for $\boldsymbol{\beta}$. We'll only compute the vector of partial derivatives at this time. We use the chain rule:

$$\begin{aligned} \underset{p\times 1}{\frac{\partial \varepsilon(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}} &= 2\underset{p\times n}{\frac{\partial(\mathbf{y} - \mathbf{X}^T\boldsymbol{\beta})}{\partial \boldsymbol{\beta}^T}}\underset{n\times 1}{(\mathbf{y} - \mathbf{X}^T\boldsymbol{\beta})} \\ &= \underset{p\times n}{-2\mathbf{X}^T}\underset{n\times 1}{(\mathbf{y} - \mathbf{X}^T\boldsymbol{\beta})}, \end{aligned} \tag{5.16}$$

since $\partial(\mathbf{y} - \mathbf{X}^T\boldsymbol{\beta})/\partial\boldsymbol{\beta}^T = -\mathbf{X}$.

5.3 Exercises

1. Verify that
$$\frac{\partial \mathbf{x}^T\mathbf{A}\mathbf{x}}{\partial \mathbf{x}} = \left(\mathbf{A}^T + \mathbf{A}\right)\mathbf{x} \tag{5.17}$$
by differentiating
$$\mathbf{x}^T\mathbf{A}\mathbf{x} = \sum_{j=1}^{p}\sum_{i=1}^{p} a_{i,j}x_i x_j. \tag{5.18}$$
with respect to $x_k, k = 1, \ldots, p$.
2. Argue that $\partial\mathbf{x}^T\mathbf{x}/\partial\mathbf{x} = 2\mathbf{x}$ by appealing to equation (5.9).
3. Determine the estimator of $\boldsymbol{\beta}$ from equation (5.16).

Chapter 6
Predictive Analytics

6.1 Introduction

The goal of predictive analytics is to predict a target vector $\mathbf{y}$ using a vector of predictor values $\mathbf{x}$ and a function $f(\mathbf{x}|D)$ that has been trained on a data set D. We suppose that the data set consists of pairs of observations on both a target value and predictor vector, say, $D = \{(\mathbf{y}_1, \mathbf{x}_1), \ldots, (\mathbf{y}_n, \mathbf{x}_n)\}$. The prediction function f is to be applied when $\mathbf{x}$ is observed but not $\mathbf{y}$. For instance, a digital image consists of a vector of pixel-based reflectance intensities $\mathbf{x}$ and the image may contain within it a facial image of a person of interest. The target, unobserved if only $\mathbf{x}$ is known, is the name of the person in the image. In this example, the target is a scalar label (her name).

We consider two types of predictive analytics (not completely distinct): regression techniques and machine learning techniques. Regression techniques are well-established and well-understood and based on statistical modeling principles. These methods tend to be simple, mathematically and computationally, as was fitting for their heyday and the practitioners. Machine learning, on the other hand, dispenses with statistical modeling principles in favor of computational brute force. The best approach depends on the specifics of the task and the available data.

Objective functions lie at the core of estimation, prediction, and forecasting methods. The objective function connects the general method, e.g., linear regression a specific predictive function. For example, $\widehat{y} = \widehat{\beta}_0 + \widehat{\beta} x_1 + \cdots + \widehat{\beta}_p x_p$ is the specific form of a regression model involving variables $x_1, \ldots, x_p$. The 'hat' above a term indicates that the term is estimated or predicted using data and a particular technique. The prediction function is used to compute a prediction ($\widehat{y}$) from an input vector $\mathbf{x} = [1 \; x_{i,1} \; x_{i,2} \; \cdots \; x_{i,p}]$. One may think of the technique as a blueprint and the specific form as the built object that may be applied to compute the target (estimates, predictions, or forecasts). The process of creating the specific form is usually called *training* when the task is prediction or forecasting, and *model fitting* in the case of

estimation. Training begins with selecting an objective function that quantitatively measures prediction error and deciding on the general form of the prediction function. The training process determines the specific form of the function that minimizes the prediction error when the rule is applied to the data. For example, if the prediction function is a k-nearest neighbor function, we'll select the neighborhood size k that yields the smallest prediction error. If the target is a quantitative variable, then one choice of objective function is the sum of the absolute error $\sum_{i=1}^{n} |y_i - \widehat{y}_i|$ ($\widehat{y}_i$ is a prediction of y_i). On the other hand, if the target is qualitative (a variable with a levels, say group identifiers), then the error function may be the sum of the *misclassified* targets: $\sum_{i=1}^{n} I_{y_i}(\widehat{y}_i)$, where $I_{y_i}(\widehat{y}_i) = 1$ if $y_i = \widehat{y}_i$ and $I_{y_i}(\widehat{y}_i) = 0$ if $y_i \neq \widehat{y}_i$.

Two methods must be developed to implement this scheme: methods of minimizing the objective function, and 2. methods of assessing error. Before proceeding, let's establish some notation and terminology.

6.1.1 Notation and terminology

A *prediction (or forecasting) function* f will be constructed from a training set. The training set is a set of pairs consisting of a predictor vector $\mathbf{x}_i$ and an observed target (either a scalar, or real number, y_i), or a vector $\mathbf{y}_i$. For now, suppose that the target is a scalar y_i. Both quantitative and qualitative targets are included. The input to the prediction function is a *predictor vector* denoted by $\mathbf{x}_t$. The data set is denoted as $D = \{(y_1, \mathbf{x}_1), \ldots, (y_i, \mathbf{x}_i), \ldots, (y_n, \mathbf{x}_n)\}$.

For example, a forecasting solution might form a predictor vector $\mathbf{x}_t$ from p past values observed on a target variable, say, the closing price of a stock, and collected as a row vector

$$\underset{1\times p}{\mathbf{x}_t} = [x_{t-p+1} \quad x_{t-p+2} \quad \cdots \quad x_t].$$

We've changed from using i as the index to t as a reminder that the data are chronologically ordered and that this ordering is important for prediction. The prediction function might be a linear predictor in which case it produces the prediction by combining the past values as a sum, say,

$$f(\mathbf{x}_t|D) = \widehat{\beta}_0 + \sum_{i=1}^{p} x_{t-p+i}\widehat{\beta}_i. \tag{6.1}$$

The coefficient vector $\widehat{\boldsymbol{\beta}} = [\widehat{\beta}_0 \quad \widehat{\beta}_1 \quad \widehat{\beta}_2 \quad \cdots \quad \widehat{\beta}_p]^T$ is determined from the training data and an objective function that measures the forecasting error.

The forecast of $y_{t+\tau}$ can be written more simply as the inner or dot product of $\mathbf{x}_t$ and $\widehat{\boldsymbol{\beta}}$:

$$\widehat{y}_t = \widehat{\beta}_0 + \underset{1\times pp\times 1}{\mathbf{x}_t\ \widehat{\boldsymbol{\beta}}}\,.$$

Usually, $\mathbf{x}_t$ is augmented with a leading one and β_0 is included in $\boldsymbol{\beta}$. Both vectors then have length $p+1$ and the forecasting equation becomes more simply $\widehat{y}_t = \mathbf{x}_t\widehat{\boldsymbol{\beta}}$.

6.1.2 Models

A model is a simple abstraction of a process or population. Herein, we consider mathematical models created for the purpose of capturing the primary features of the process and thereby gaining insight to process, or for building a mathematical object that approximates the process.[1] Traditional statistics are strongly dependent on models, and in some sense, the model motivates the methods used by an analyst. Recent advances in machine learning notably do not attempt to realistically model the process of interest but instead use massively large datasets and computational power to build what might best be thought of as a large mathematical approximation function (e.g., automatic language translation). With this in mind, the term *model* should not be used for all the cases of interest since there are methodologies that purposely avoid positing a model. However, there isn't a clear split between model-based and model-free methods. For example, k-nearest neighbor prediction functions and artificial neural networks use what is usually called a model without any effort at realistically creating a simple abstraction of the underlying process that has generated the training data. Often, a prediction function is called a model out of convenience and not because it provides insight to the underlying process.

If the objectives are principally prediction or forecasting, then the model, if there is one, is only a stepping stone used to achieve the objective of accurate predictions. The model itself is often uninteresting (a justifiable circumstance if the process is so complicated that forming a realistic model is unrealistic).[2] Our objectives are principally prediction and forecasting, and so models are little more than conveniences and not of intrinsic interest.

The application of estimation, prediction, and forecasting methods within the realm of predictive analytics is guided by the desire to produce accurate estimates, predictions, and forecasts. Doing so requires examples—data—

[1] More specifically, the mathematical object is a function or algorithm that generates estimates, predictions, or forecasts.

[2] For example, it's inconceivable that a realistic model can be developed for the application of forecasting the national deficit given the large number of interacting factors that affect the debt. The situation is a little different when the objective is estimation because estimation usually attempts to characterize a population or process and a model in essence, represents or characterizes the population or process. Estimation is is a central theme of statistics. Accurate estimation requires both approximately realistic model and representative data.

with which accuracy may be estimated and the method may be trained to yield accurate estimates, predictions, and forecasts. Naturally, the data ought to be representative of the process or population to which the method will be applied.[3] This need will lead us to cross-validation, a generally reliable and easy to implement approach of accuracy estimation

We will discuss several important prediction methods that are model-based in the following sections. Keep in mind that the models are not constructed to be realistic representations of the populations or processes from which the targets of prediction and forecasting originate, but more simply, mathematical approximations of the targets.

6.2 Regression Techniques

We focus on two type of models and regression methods: linear models least squares, and nonlinear models and maximum likelihood.

6.2.1 Linear models and least squares

The linear model prediction function is

$$\widehat{y} = \beta_0 + \sum_{i=1}^{p} \beta_i \mathbf{x}_i. \tag{6.2}$$

Using vector notation, $\widehat{y} = \mathbf{x}\boldsymbol{\beta}$ after having set

$$\underset{1\times q}{\mathbf{x}} = [1 \;\; x_1 \;\; x_2 \;\; \cdots \;\; x_p],$$

$\boldsymbol{\beta} = [\beta_0 \;\; \beta_1 \;\; \cdots \;\; \beta_p]^T$ and $q = p + 1$. We've departed slightly from the notation used in the earlier example (equation 6.1.1). The least squares objective function is most often used for linear models , and it is

$$\varepsilon(\boldsymbol{\beta}) = \sum_{i=1}^{n} (y_i - \mathbf{x}_i\boldsymbol{\beta})^2, \tag{6.3}$$

where $(y_i, \mathbf{x}_i) \in D$ is an observed observation pair. In matrix form, the objective function is

$$\varepsilon(\boldsymbol{\beta}) = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}), \tag{6.4}$$

[3] Creating a representative data set is a central theme of statistics. We'll proceed under the assumption that the data is representative and focus on the problem of achieving maximal accuracy.

where $\mathbf{y}$ is an n-vector of observed target values and $\mathbf{X}$ is $n \times q$ and constructed by stacking the n predictor vectors as rows. The least squares estimator of $\boldsymbol{\beta}$ is determined by differentiating $f(\boldsymbol{\beta})$ with respect to $\boldsymbol{\beta}$, setting the vector of derivatives equal to $\mathbf{0}$, and solving for $\boldsymbol{\beta}$. This process yields the least squares estimator

$$\widehat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{6.5}$$

Equation (6.5) defines the best possible coefficient vector in terms of minimizing the the sum of the squared errors when the prediction function is applied to the data set. It's important to note that the data has been used twice, in a somewhat abstract sense: once to determine the coefficients and a second time to assess the error. A completely distinct *test* data set might be used to assess the error and would surely yield a different sum of squared errors (even if the number of pairs in test set were the same). If both sets were representatively drawn from the same population, the training set estimate of error usually will be smaller than the test set. This phenomenon, known as *over-fitting* will be explored in detail later, but for now, we often can improve the prediction error when the prediction function is applied to data sets besides the training set by introducing a regularization term into the original objective function.[4] The revised objective function is

$$\begin{aligned} \varepsilon_R(\boldsymbol{\beta}) &= (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda||\boldsymbol{\beta}||_2^2 \\ &= (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda(\boldsymbol{\beta}^T\boldsymbol{\beta}), \end{aligned} \tag{6.6}$$

where $||\boldsymbol{\beta}||_2$ is the L_2, or Euclidean norm of the vector $\boldsymbol{\beta}$. It's relatively easy to determine the regularized coefficient vector by solving the matrix equation $\partial\varepsilon_R(\boldsymbol{\beta})/\partial\boldsymbol{\beta} = \mathbf{0}$ for $\boldsymbol{\beta}$. The solution is

$$\widehat{\boldsymbol{\beta}}_R = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_p)^{-1}\mathbf{X}^T\mathbf{y}. \tag{6.7}$$

6.2.2 Least absolute deviation regression and the lasso

Let us consider modifying the least squares objective function. The change is to replace the sum of the squared errors by the sum of the absolute errors. The difference between the two objective functions lies with the effect of a particular difference $y_i - \widehat{y}_i$ on the objective function. With regard to the absolute error objective function, the influence of an error equal to η in magnitude on the objective function is half of the effect of an absolute error of 2η. If the objective function is the sum of the squared errors, then the effect of an absolute error equal to 2η is instead $4\eta^2$. Consequently, large-magnitude errors have greater influence on the least squares estimator than the absolute

[4] The consequence of introducing the regularization term is to reduce the norm of the coefficient vector.

error function. Using the sum of the absolute errors is referred to as least absolute deviation regression and, alternatively, as L_1 regression because the objective function is the L_1 norm of the vector of deviations $\mathbf{y} - \widehat{\mathbf{y}}$.

The absolute errors objective function is

$$\begin{aligned} \varepsilon(\boldsymbol{\beta}) &= \sum_{i=1}^{n} |y_i - \mathbf{x}_i\boldsymbol{\beta}| \\ &= \mathbf{j}^T|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}|, \end{aligned} \tag{6.8}$$

where $\mathbf{j}$ is a n-vector of 1's and $|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}| = [|y_1 - \mathbf{x}_1\boldsymbol{\beta}| \ \cdots \ |y_n - \mathbf{x}_n\boldsymbol{\beta}|]^T$

Introducing a regularization term into the objective function not only helps reduce prediction error but also may used for variable selection, in which case, the method is referred to as the lasso.[5][**?**] The regularization term is not quite the same as in equation (6.6) because the L_2 norm of $\boldsymbol{\beta}$ is replaced with the L_1 norm. The objective function is therefore

$$\begin{aligned} \varepsilon(\boldsymbol{\beta}) &= \sum_{i=1}^{n} |y_i - \mathbf{x}_i\boldsymbol{\beta}| + \lambda \sum_{i=0}^{p} |\beta_i| \\ &= \mathbf{j}^T|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}| + \lambda||\boldsymbol{\beta}||_1, \end{aligned} \tag{6.9}$$

where$||\boldsymbol{\beta}||_1$ is the L_1 norm of $\boldsymbol{\beta}$. At the present time, our interest is in the computational challenge of minimizing the objective function.

We may follow the approach used in least squares regression—differentiate the objective function with respect to $\boldsymbol{\beta}$, set the vector of partial derivatives equal to the zero vector and attempt to solve the matrix equation for $\boldsymbol{\beta}$. It that can be accomplished, then a closed-form, or analytical form will have been determined for the estimator of $\boldsymbol{\beta}$.

First, the absolute value function is differentiable everywhere but 0, and for $x \neq 0$,

$$\begin{aligned} \frac{\mathrm{d}|x|}{\mathrm{d}x} &= \begin{cases} -1, & \texttt{if } x < 0, \\ 1, & \texttt{if } x > 0 \end{cases} \\ &= I_+(x) - I_+(-x), \end{aligned} \tag{6.10}$$

where

$$I_+(x) = \begin{cases} 1, & \texttt{if } x > 0 \\ 0, & \texttt{if } x \leq 0 \end{cases}$$

If we ignore the possibility that one of the β_i's may be zero, $\partial\varepsilon(\boldsymbol{\beta})/\partial\boldsymbol{\beta}$ can be computed and set to $\mathbf{0}$. However, an analytical (closed-form) solution cannot be found.[6] The minimizing value $\widehat{\boldsymbol{\beta}}$ will have to be found using an alternative

[5] lasso is an acronym for *least absolute shrinkage and selection operator*.

[6] If there were an analytical solution, then the median of a set of observations could be computed using a formula. This, of course not the case, and we must resort to a sorting or sorting-like algorithm to find the median.

approach. The next section discusses a simple, reliable, and slow method of determining the minimizing value of $\boldsymbol{\beta}$.

6.2.3 Pattern search

Pattern search (or direct search) is an iterative algorithm that does not utilize the derivatives of the objective function. In brief, it begins with an arbitrary initial value for $\boldsymbol{\beta}$. Then, an iteration phase commences and on each iteration, each coefficient is perturbed by adding or subtracting a constant. It the modified vector yields a smaller value for the objective function, then the coefficient vectors is updated (the perturbation, say β_i is replaced by $\beta_i + \epsilon$, otherwise, the coefficient vector is not changed). If, after cycling through each coefficient, there's be no update, then the perturbation is made smaller (halved, say) and the next iteration repeats the process of cycling through each coefficient with the smaller perturbations. There's no guarantee that the algorithm will succeed at finding the minimum value, but if it is used with care, it probably will produce a coefficient vector very near the minimizing value $\widehat{\boldsymbol{\beta}}$.

The steps to building and implementing a pattern search algorithm are described in more detail:

1. Create a function $\varepsilon(\mathbf{y}, \widehat{\mathbf{y}})$ that computes the objective function from the actual and predicted target vectors.
2. Set the number of iterations, say $n_I = 100$, and the initial perturbation, say $\eta = 1$.
3. Initialize $\boldsymbol{\beta}$, say $\boldsymbol{\beta}_0 = \mathbf{0}$.
4. Compute $\widehat{\mathbf{y}}$ using $\boldsymbol{\beta}_0$ and then $\varepsilon(\mathbf{y}, \widehat{\mathbf{y}})$.
5. Begin iterating. On each iteration (indexed by i),
    a. Create a copy[7] of $\boldsymbol{\beta}$, say $\boldsymbol{\beta} \rightarrow \boldsymbol{\beta}_i$.
    b. Cycle through each coefficient $\beta_{i,0}, \beta_{i,1}, \ldots, \beta_{i,p}$ and do the following:
        i. Perturb $\beta_{i,j}$ by computing $\beta_{i,j} \leftarrow \beta_{i,j} + \eta$ where $\beta_{i,j}$ is the current value of the jth coefficient.
        ii. Compute $\widehat{\mathbf{y}}$ using $\boldsymbol{\beta}_i$.
        iii. Compute $\varepsilon(\mathbf{y}, \widehat{\mathbf{y}})$.
        iv. If $\varepsilon(\mathbf{y}, \widehat{\mathbf{y}})$ has *not* been reduced relative the the previous value of $\varepsilon(\mathbf{y}, \widehat{\mathbf{y}})$, then reset the coefficient by computing $\boldsymbol{\beta}_{i,j} - \eta \rightarrow \boldsymbol{\beta}_{i,j}$.
        v. Cycle through each coefficient $\beta_0, \beta_1, \ldots, \beta_p$ but *subtract* η instead of adding η. Carry out steps 5(b)ii, 5(b)iii and 5(b)iv except that resetting the coefficient (when necessary) is accomplished by computing $\boldsymbol{\beta}_{i,j} + \eta \rightarrow \boldsymbol{\beta}_{i,j}$.

[7] We use the arrow to indicate that the value of $\boldsymbol{\beta}$ is assigned to $\boldsymbol{\beta}_i$.

c. If the objective function was not reduced in step 5b, then reduce η by a factor between 0 and 1, say, $\eta \leftarrow .7\eta$. If there was a reduction in the objective function, then do not change η.
d. Print the iteration counter, $\varepsilon(\mathbf{y}, \widehat{\mathbf{y}})$, η, and a boolean variable indicating whether or the objective function was reduced during the current iteration.
e. Begin the next iteration (return to step 5a).

Pattern search will always produce a coefficient vector that is near the true optimal vector if the objective function has no local minimums that can trap the algorithm. There are numerous other more sophisticated algorithms for finding optimal vectors. Let's turn now to a faster algorithm that works when the objective function is well-behaved.

6.3 Newton's method

Newtons' method is a far more efficient means of optimizing an objective function than pattern search. There are limitations on it's utility since it can be used only if the function is differentiable. If the function of interest is an objective function, then it is also necessary that the matrix of second derivatives, say $\partial\varepsilon^2(\boldsymbol{\beta})/\partial\boldsymbol{\beta}\partial\boldsymbol{\beta}^T$ can be computed.[8] We begin the discussion assuming that the function of interest is univariate, differentiable, and defined on $\mathbb{R}$.

A function f that is differentiable on the interval $[a, b]$ has a derivative at every $x \in [a, b]$.[9] This condition is satisfied often enough by commonly used objective functions that Newton's method is useful algorithm for predictive analytics. In fact, it's widely used for maximum likelihood estimation, including logistic regression and other *generalized linear models*. In most applications, the objective function f is defined for every real number in an interval of interest. For this discussion, let's assume that f has a unique global minimum at x^*,[10] and that the objective is to determine x^*. Since f is differentiable, the derivative of f at $x_0 \in \mathbb{R}$ is the limit[11]

[8] The matrix of second derivative is referred to as the Jacobian of f and sometimes the *Hessian* matrix.

[9] In other words, $\mathrm{d}f(y)/\mathrm{d}y|_x$ exists for every $x \in [a, b]$.

[10] Local minima are not uncommon in predictive analytics.

[11] Mathematician rarely compute derivatives by directly computing the limit shown in equation (6.12) because there are a handful of rules for computing derivatives that are easier to apply. For the sake of understanding derivatives, though, let's compute the derivative of $f(x) = x^2$. First, let's change the expression slightly by setting $\epsilon = x - x_0$. Then,

$$f'(x_0) = \lim_{\epsilon \to 0} \frac{f(x_0 + \epsilon) - f(x_0)}{\epsilon}.$$

Now replace $f(x_0)$ and $f(x_0 + \epsilon)$ with x_0^2 and $(x_0 + \epsilon)^2 = x_0^2 + 2x_0\epsilon + \epsilon^2$:

$$f'(x_0) = \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0} \tag{6.12}$$

For the reader that is not familiar with calculus, the fraction in equation (6.12) is equivalent to the slope of a line $\{(x, f(x))|x \in \mathbb{R}\}$ assuming that f is a linear function. Therefore, the derivative of a function f at x_0 may be thought of as the slope of a linear approximation to f made at x_0.

If x is close to x_0, then we can approximate $f(x_0)$ rearranging the approximation

$$f'(x_0) \approx \frac{f(x) - f(x_0)}{x - x_0} \tag{6.13}$$

to obtain the Taylor series first-order approximation of f at x_0:

$$f(x_0) \approx f(x) + f'(x)(x_0 - x). \tag{6.14}$$

Equation (6.14) is a *linear approximation* since $f(x) - f'(x)x$ is constant and not depending on the variable x_0, and the variable x_0 is multiplied by a coefficient $f'(x)$. The sum of the two terms describes as linear function. The approximation also can be rearranged again to yield an approximation of x that satisfies an equation, say $y = f(x)$ providing that x_0 is close to x and the values $f(x_0)$ and $f'(x_0)$ are known. To do so, replace $f(x)$ with the value y and choose x_0 near x. The approximation x_1 of x is

$$x_1 = x_0 + \frac{f(x) - f(x_0)}{f'(x_0)} = x_0 + \frac{y - f(x_0)}{f'(x_0)}. \tag{6.15}$$

The accuracy of the approximation is depends on how far x_0 is from x (closer is better of course) and the degree of linearity of f at x_0. Newton's method successively computes improved approximations of x.

Specifically, an algorithm implementing Newton's method begins with a calculation of x_1 according to equation (6.15) and an initial guess x_0. Then, iterations commence using the iteration counter k and on each iteration, the

---

$$\begin{aligned} f'(x_0) &= \lim_{\epsilon \to 0} \frac{x_0^2 + 2x\epsilon + \epsilon^2 - x_0^2}{\epsilon} \\ &= \lim_{\epsilon \to 0} \frac{2x_0\epsilon + \epsilon^2}{\epsilon} \\ &= 2x_0 + \lim_{\epsilon \to 0} \epsilon = 2x_0. \end{aligned} \tag{6.11}$$

Now, we must imagine a number ϵ that's positive to begin with and shrinks to 0. For example, imagine a series of infinitely many values and the ith value is the previous value divided by 2. The limit $\lim_{\epsilon \to 0} \epsilon$ is the outcome of this mental exercise, and for all practical purposes, the outcome is 0. The argument justifying this statement use the fact that the limit is positive because any positive number divided by 2 is positive. Now, if you propose any nonzero number to be the limit, there is a step at which the value of ϵ is smaller (by going sufficiently far into the series) and so that value is better approximation of the limit. As there is no smallest number greater than zero, zero is the limit, and we write $\lim_{\epsilon \to 0} \epsilon = 0$. Returning to the last equation, we deduce that $f'(x_0) = 2x_0$.

previous value x_{k-1} is replaced with the most recently computed x_k. So, on iteration k, compute

$$x_k = x_{k-1} + \frac{y - f(x_{k-1})}{f'(x_{k-1})}. \tag{6.16}$$

As $f(x_{k-1})$ approaches y, $y - f(x_{k-1})$ tends to zero and the adjustment to the previous approximation tends to zero.

For many functions, Newton's method will not only converge to the solution, but the rate of convergence very rapid. For many problems, the convergence rate is quadratic and so as iterations take place, the difference between the solution and the approximation is a power of 2.[12]

The first-order Taylor series approximation of a multivariate function $f : \mathbb{R}^p \to \mathbb{R}^q$ is

$$\underset{q\times 1}{f(\boldsymbol{\beta}_0)} \approx \underset{q\times 1}{f(\boldsymbol{\beta})} + \underset{q\times p}{\frac{\partial f(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}^T}}(\underset{p\times 1}{\boldsymbol{\beta}_0} - \boldsymbol{\beta}). \tag{6.17}$$

6.3.1 Applying Newton's method to objective functions

We consider an objective function ε that is differentiable and maps $\mathbb{R}^p$ to $\mathbb{R}^+$, the non-negative real numbers, for some non-negative integer p. We'll use $\boldsymbol{\beta}$ as the argument; hence, $\varepsilon : \boldsymbol{\beta} \mapsto c \in \mathbb{R}^+$. Newton's method will be used to determine $\widehat{\boldsymbol{\beta}}$ that minimizes ε. To use Newton's method as it has been described requires the value $\varepsilon(\widehat{\boldsymbol{\beta}})$, but this value is unknown until $\widehat{\boldsymbol{\beta}}$ has been determined. To get around this problem, we utilize the fact that $\partial\varepsilon/\partial\boldsymbol{\beta}|_{\widehat{\boldsymbol{\beta}}} = \mathbf{0}$, since $\widehat{\boldsymbol{\beta}}$ minimizes $\varepsilon(\cdot)$. Therefore, we'll solve the equation $\partial\varepsilon(\widehat{\boldsymbol{\beta}})/\partial\boldsymbol{\beta} = \mathbf{0}$ for $\boldsymbol{\beta}$ using Newton's method.

Before proceeding to the multivariate case, let us develop the univariate case ($p = 1$). Using equation (6.16) and setting $y = 0$ leads to the updating formula for the kth iteration:

$$\beta_k = \beta_{k-1} - \frac{\varepsilon'(\beta_{k-1})}{\varepsilon''(\beta_{k-1})}, \tag{6.18}$$

for $\varepsilon''(\beta_{k-1}) \neq 0$.

Now consider a p-vector $\boldsymbol{\beta}$, for $p > 1$ and the problem of finding $\widehat{\boldsymbol{\beta}}$ that minimizes $\varepsilon(\boldsymbol{\beta})$. In general, $\varepsilon(\boldsymbol{\beta})$ is unknown, but if ε is differentiable, then it is known that $\partial\varepsilon(\widehat{\boldsymbol{\beta}})/\partial\boldsymbol{\beta} = \mathbf{0}$ and Newton's method can be used to solve the matrix equation

$$\mathbf{0} = \frac{\partial\varepsilon(\widehat{\boldsymbol{\beta}})}{\partial\boldsymbol{\beta}} = f(\widehat{\boldsymbol{\beta}}) \tag{6.19}$$

[12] The difference would follow a sequence such as $10^{-2}, 10^{-4}, 10^{-8}, \ldots$.

for $\widehat{\boldsymbol{\beta}}$. Since $\widehat{\boldsymbol{\beta}}$ minimizes $\varepsilon(\boldsymbol{\beta})$ with respect to $\boldsymbol{\beta}$, $\partial\varepsilon(\widehat{\boldsymbol{\beta}})/\partial\boldsymbol{\beta} = \mathbf{0}$, and the first-order Taylor series expansion of $\varepsilon : \mathbb{R}^p \to \mathbb{R}^p$ about $\widehat{\boldsymbol{\beta}}$ is

$$\underset{p\times 1}{\mathbf{0}} = \underset{p\times 1}{\frac{\partial\varepsilon(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}}} + \underset{p\times p}{\frac{\partial\varepsilon^2(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}\partial\boldsymbol{\beta}^T}}\underset{p\times 1}{(\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta})}. \tag{6.20}$$

Re-arranging equation (6.20) provides the updating formula for determining $\widehat{\boldsymbol{\beta}}$ by Newton's method:

$$\boldsymbol{\beta}_k = \boldsymbol{\beta}_{k-1} - \left(\frac{\partial\varepsilon^2(\boldsymbol{\beta}_{k-1})}{\partial\boldsymbol{\beta}\partial\boldsymbol{\beta}^T}\right)^{-1}\frac{\partial\varepsilon(\boldsymbol{\beta}_{k-1})}{\partial\boldsymbol{\beta}}. \tag{6.21}$$

Obviously, the inverse of the matrix of second derivatives (known as the *Hessian* matrix) must exist to use Newton's method.

6.4 Exercises

1. Compute $\mathrm{d}x^{-1}/\mathrm{d}x$ using a limit definition of the derivative (formula 6.12.)
2. Provide the `Python` code for finding a solution to the equation $f(x) = x^2 - 2$ using Newton's method or pattern search. (This is a clever way to compute $\sqrt{2}$ if your smart phone needs to be recharged).
3. Provide the `Python` code (and the solution) for solving the equation $1 = e^x - x$ for x using Newton's method or pattern search.
4. Provide the `Python` code for finding all of the roots of the polynomial $g : [-5, 20] \to \mathbb{R}$ given by

$$g(x) = 30 + .1x - 3x^2 + .2x^3 - .001x^4. \tag{6.22}$$

   Hint: graph the function so that you can visually identify initial values for solving for each of the roots. What are the roots?
5. Suppose that the objective function of interest is by equation (6.4). Derive the updating formula for applying Newton's method to the computation of $\widehat{\boldsymbol{\beta}}$, thereby showing that only a single iteration is necessary to arrive at the minimizing value of $\boldsymbol{\beta}$.
6. Suppose that the objective function is the *cross-entropy* function

$$\varepsilon(\boldsymbol{\beta}) = -\sum_{i=1}^{n} y_i\mathbf{x}_i\boldsymbol{\beta} + \sum_{i=1}^{n}\log\left[1 + \exp(\mathbf{x}_i\boldsymbol{\beta})\right], \tag{6.23}$$

where $\mathbf{x}_i$ is $1 \times q$ and $\boldsymbol{\beta}$ is $q \times 1$. Equation (6.23) is the objective function used for logistic regression.[13] Verify the updating formula for Newton's method applied to equation (6.23) is

$$\underset{q\times 1}{\boldsymbol{\beta}_k} = \underset{q\times 1}{\boldsymbol{\beta}_{k-1}} + (\underset{q\times q}{\mathbf{X}^T\mathbf{W}_{k-1}\mathbf{X}})^{-1}\underset{q\times n}{\mathbf{X}^T}[\underset{n\times 1}{\mathbf{y}} - \underset{n\times 1}{\boldsymbol{\pi}(\boldsymbol{\beta}_{k-1})}], \tag{6.24}$$

where

$$\pi_i(\boldsymbol{\beta}_{k-1}) = [1 + \exp(-\mathbf{x}_i\boldsymbol{\beta}_{k-1})]^{-1}$$
$$\boldsymbol{\pi}(\boldsymbol{\beta}_{k-1}) = [\pi_1(\boldsymbol{\beta}_{k-1}) \;\; \pi_2(\boldsymbol{\beta}_{k-1}) \;\; \cdots \;\; \pi_n(\boldsymbol{\beta}_{k-1})],$$

and

$$\underset{n\times n}{\mathbf{W}_{k-1}} = \text{diag}\{\pi_i(\boldsymbol{\beta}_{k-1})[1 - \pi_i(\boldsymbol{\beta}_{k-1})]\}. \tag{6.25}$$

7. Write a function that carries out pattern search optimization. (See section 6.2.3 above). Compute the least squares estimator using the Boston housing data[14]. The data and a `Python` function to read the data are posted on the Moodle website in the *Introduction to Predictive Analytics* topic section. The target variable is `medv`—the median value of owner-occupied homes in $1000's. Use all other 12 variables as predictors, and as usual, the constant 1.

   a. Compute the least squares estimator of $\boldsymbol{\beta}$ directly[15] and using your pattern search algorithm. Verify that the two methods yield coefficient vectors that are essentially the same.
   b. Compute the coefficient vector using regularized least squares. (You will have to modify the objective function to include the penalty term). It may be helpful to verify that the usual least squares estimator is computed when $\lambda = 0$. Then set $\lambda = .001$.

8. Write a Newton's method algorithm for logistic regression. It's assumed that a design matrix $\mathbf{X}$ and target vector $\mathbf{y}$ have been constructed as `Numpy` matrices of dimension $n \times q$ and $n \times 1$, respectively. Presumably, one of the columns of $\mathbf{X}$ consists of ones.

   a. Write a function that computes the log-likelihood function (equation 7.4). The arguments passed to the function are $\mathbf{X}$ and $\mathbf{y}$, and a coefficient vector $\boldsymbol{\beta}$, specifically, the kth iteration estimate of $\boldsymbol{\beta}$. The function returns the log-likelihood evaluated at the input vector $\boldsymbol{\beta}$.
   b. Create the initial estimate of $\boldsymbol{\beta}$ as a `Numpy` matrix, say

[13] Cross-entropy is the negative of the log-likelihood, and so $\widehat{\boldsymbol{\beta}}$ can be determined by *minimizing* equation 6.23 with respect to $\boldsymbol{\beta}$

[14] Details: `https://www.kaggle.com/c/boston-housing`

[15] Specifically, compute $\widehat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$.

```
b = np.matrix(np.zeros(shape = (q, 1)))
```

c. Create a `while` loop that iterates until the change between successive evaluations of the objective function is smaller than some ϵ. The shell of the loop is:

```
k = 0
while (abs(previousValue - objFnValue)) > 1e-4:
    previousValue = objFnValue
    ...
    objFnValue = objFn(X, Y, b)
    print(k, objFnValue)
    k += 1
```

On the kth iteration, $k = 1, 2, \ldots$, Carry out the following operations:

d. Compute $\pi(\boldsymbol{\beta}_k)$ according to

$$\begin{aligned} \pi_i(\boldsymbol{\beta}_{k-1}) &= [1 + \exp(-\mathbf{x}_i\boldsymbol{\beta}_{k-1})]^{-1}, \\ \boldsymbol{\pi}(\boldsymbol{\beta}_{k-1}) &= [\pi_1(\boldsymbol{\beta}_{k-1}) \;\; \pi_2(\boldsymbol{\beta}_{k-1}) \;\; \cdots \;\; \pi_n(\boldsymbol{\beta}_{k-1})]. \end{aligned} \tag{6.26}$$

e. Compute the product $\mathbf{W}_k\mathbf{X}$. A computationally efficient method initializes $\mathbf{W}_k$ as a copy of $\mathbf{X}$, say, `WX = X.copy()`. Then, iterate over the rows of `WX` and replace ith row by the product of $\mathbf{x}_i$ and $w_{ii} = \pi_i(\boldsymbol{\beta}_k)[1 - \pi_i(\boldsymbol{\beta}_k)]$, say,

```
for i, x in enumerate(WX):
    WX[i,:] *= float((1 - piVector[i])*piVector[i])
```

The operation `a *= b` multiplies `a` by `b` and assigns the product to `a`.

f. Compute $\mathbf{X}^T\mathbf{W}_k\mathbf{X}$ and at the same time update $\boldsymbol{\beta}_k$ according to

$$\underset{q\times 1}{\boldsymbol{\beta}_k} = \underset{q\times 1}{\boldsymbol{\beta}_{k-1}} + (\underset{q\times q}{\mathbf{X}^T\mathbf{W}_{k-1}\mathbf{X}})^{-1}\underset{q\times n}{\mathbf{X}^T}[\underset{n\times 1}{\mathbf{y}} - \underset{n\times 1}{\boldsymbol{\pi}(\boldsymbol{\beta}_{k-1})}] : \tag{6.27}$$

```
b += np.linalg.solve(X.T*WX, X.T*(Y - piVector))
```

g. Compute the objective function and increment k. Program flow must then return to the beginning of the `while` loop so that the test (`(abs(previousValue - objFnValue)) > 1e-4`) is evaluated.

h. Complete the function by returning $\boldsymbol{\beta}_k$.

i. It's helpful to print the iteration counter and the objective function on each iteration (at least while coding the function).

j. Apply the estimation function to the Wisconsin breast cancer data set to compute $\widehat{\boldsymbol{\beta}}$.[16] The data, and a `Python` function to read the data file, build the design matrix $\mathbf{X}$, and the target vector $\mathbf{y}$ are posted on Moodle.
It's useful to note that ith row of $\underset{n\times q}{\mathbf{W}_{k-1}\mathbf{X}}$ is $[x_{i,1}w_i \;\; x_{i,2}w_i \;\; \cdots \;\; x_{i,q}w_i]$.
It's probably easier and more efficient to compute $\mathbf{W}_{k-1}\mathbf{X}$ by row rather than forming the diagonal matrix $\mathbf{W}$ and then computing the product of $\mathbf{W}$ and $\mathbf{X}$.

k. Report the *apparent accuracy* given by the proportion of predictions $\widehat{y}_i, i = 1, 2, \ldots, n$ that are equal to the actual values $y_i, i = 1, 2, \ldots, n$.

[16] `https://www.kaggle.com/uciml/breast-cancer-wisconsin-data`

Chapter 7
Qualitative Targets

7.1 Introduction

Two related model-based predictive methods are discussed in this chapter as well as several methods used for accuracy assessment.

7.2 Logistic regression

Logistic regression has two primary purposes: prediction, and regression analysis. Regression analysis is a broad class of methods aimed at developing inferences about populations or processes with the ultimate goal of making statistically defensible statements about the population or process of interest. We will focus on the narrower task of predicting an unobserved *qualitative* target value. In the case of logistic regression, the qualitative variable identifies membership in one of two classes, and so the qualitative variable is also a binary variable. As usual, the observations are pairs $\mathbf{z} = (y, \mathbf{x})$ where y is the identifier of membership and $\mathbf{x}$ is the target. In this context, it's useful to view $(y, \mathbf{x})$ as observed on a unit (or person). For example, y may identify whether or not an attachment to an email is malware or not. The unit is the email and the predictor vector consists of information gleaned from the email sender and the text content of the email.

Let's suppose that there are two classes C_0 and C_1, and that $\mathbf{z}$ belongs to one or the other but not both. We'll suppose further that y is defined according to

$$y = \begin{cases} 1, & \mathbf{z} \in C_0, \\ 0, & \mathbf{z} \in C_1. \end{cases}$$

The task at hand is this: if $\mathbf{x}_0$ observed, but not y_0, predict y_0 from $\mathbf{x}_0$ using a function $f(\cdot|D)$. All of the targets y_i, for $\mathbf{z}_i \in D$, have been observed. We will

construct f by training a logistic regression model on the data set D. However, one may bypass the justification argument of the logistic regression model and simply decide to use *cross-entropy* as the objective function, determine the coefficient vector $\widehat{\boldsymbol{\beta}}$ that minimizes the objective function, and use $\widehat{\boldsymbol{\beta}}$ and a simple decision rule to produce the prediction $\widehat{y}_0$.

We'll take the longer approach to developing the predictive function (through) maximum likelihood since there are other predictive functions that can be developed using a maximum likelihood approach. Therefore, a probabilistic view is adopted and each y is presumed to be the realization of a random variable Y with the following Bernoulli probability distribution:

$$\Pr(Y = y) = \begin{cases} \pi^y (1-\pi)^{(1-y)}, & \text{if } y \in \{0, 1\} \\ 0, & \text{otherwise.} \end{cases}$$

where $0 < \pi < 1$ is the probability of the event $\{Y = 1\}$. The expected value and variance of Y are $\mu = \pi$ and $\sigma^2 = \pi(1-\pi)$. To reflect the role of $\mathbf{x}$ in determining π, we'll write $\pi_i = \mu(\mathbf{x}_i)$, $\pi_0 = \mu(\mathbf{x}_0)$, or simply or $\pi = \mu(\mathbf{x})$.

A regression model of $\pi(\mathbf{x})$ for a Bernoulli random variable should not produce fitted values or predictions that are outside the domain of π (specifically, the interval $(0, 1)$). If the model is linear and so takes the form $\pi(\mathbf{x}) = \mathbf{x}\boldsymbol{\beta}$ where $\boldsymbol{\beta} \in \mathbb{R}^q$, then the model will yield predictions outside of the unit interval. The nearly ubiquitous solution to this difficulty is to model the logit transformation of π since the logit of π may take on any real number

The logit function of π is

$$\text{logit}(\pi) = \log\left(\frac{\pi}{1-\pi}\right). \tag{7.1}$$

In following development, $\text{logit}[\pi(\mathbf{x})]$ is modeled as a linear function of $x_1, \ldots, x_p$ and the *link* between the linear model $\mathbf{x}\boldsymbol{\beta}$ and $\pi = \mu(\mathbf{x})$ is $\text{logit}[\pi(\mathbf{x})] = \mathbf{x}\boldsymbol{\beta}$.

7.2.1 The maximum likelihood estimator

As a reminder of the set-up, let us begin again. Suppose that the training data set is $D = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n\}$ where $\mathbf{z}_i = (y_i, \mathbf{x}_i)$. The target values $y_1, y_2, \ldots, y_n$ are realizations of the independent Bernoulli random variables $Y_1, Y_2, \ldots, Y_n$. For each Y_i, $\pi_i = \mu(\mathbf{x}_i) = \mathbf{x}_i\boldsymbol{\beta}$.

The likelihood function $L(\boldsymbol{\beta}|D)$ is proportional to the joint probability function of $Y_1, Y_2, \ldots, Y_n$ (proportional in the sense that terms that *do not* depend on $\boldsymbol{\beta}$ may be omitted.) By independence,

$$\begin{aligned} L(\boldsymbol{\beta}|D) &= \prod_{i=1}^{n} \Pr(Y_i = y_i|\mathbf{x}_i) \\ &= \prod_{i=1}^{n} \pi(\mathbf{x}_i)^{y_i}[1-\pi(\mathbf{x}_i)]^{(1-y_i)} \\ &= \prod_{i=1}^{n} \left[\frac{\pi(\mathbf{x}_i)}{1-\pi(\mathbf{x}_i)}\right]^{y_i} [1-\pi(\mathbf{x}_i)]. \end{aligned} \tag{7.2}$$

It's more convenient at this stage to work with the log-likelihood function. In other words, $\log[L(\boldsymbol{\beta}|D)]$ becomes the objective function and we'll determine $\boldsymbol{\beta}$ that maximizes $\log[L(\boldsymbol{\beta}|D)]$. Since the natural log function is monotonically increasing, the maximizing value of $l(\boldsymbol{\beta}|D)] = \log[L(\boldsymbol{\beta}|D)]$ is also the maximizing value of $L(\boldsymbol{\beta}|D)$.

The next step changes the parameterization using $\pi(\mathbf{x}_i), i = 1, 2, \ldots, n$, to a parameterization involving $\boldsymbol{\beta}$ exclusively. First, exercise (1) asks the reader to prove

$$\log[1-\pi(\mathbf{x}_i)] = -\log\{[1+\exp(\mathbf{x}_i\boldsymbol{\beta})]\}, \tag{7.3}$$

and we use this result to develop the log-likelihood:

$$\begin{aligned} l(\boldsymbol{\beta}|D) &= \sum_{i=1}^{n} \log \Pr(Y_i = y_i|\mathbf{x}_i) \\ &= \sum_{i=1}^{n} y_i \text{logit}[\pi(\mathbf{x}_i)] + \log[1-\pi(\mathbf{x}_i)] \\ &= \sum_{i=1}^{n} y_i\mathbf{x}_i\boldsymbol{\beta} - \log\{[1+\exp(\mathbf{x}_i\boldsymbol{\beta})]\}. \end{aligned} \tag{7.4}$$

The last step is justified in exercise (1c).

Because the maximizing value $\widehat{\boldsymbol{\beta}}$ satisfies

$$\mathbf{0} = \left.\frac{\partial l(\boldsymbol{\beta}|D)}{\partial \boldsymbol{\beta}}\right|_{\widehat{\boldsymbol{\beta}}}, \tag{7.5}$$

let's compute the gradient of l with respect to $\boldsymbol{\beta}$. We can go only so far towards determining $\widehat{\boldsymbol{\beta}}$:

$$\begin{aligned} \frac{\partial l(\boldsymbol{\beta}|D)}{\partial \boldsymbol{\beta}} &= \sum_{i=1}^{n} y_i\mathbf{x}_i^T - \frac{\exp(\mathbf{x}_i\boldsymbol{\beta})}{[1+\exp(\mathbf{x}_i\boldsymbol{\beta})]}\mathbf{x}_i^T \\ &= \sum_{i=1}^{n} [y_i - \pi(\mathbf{x}_i)]\,\mathbf{x}_i^T \\ &= \mathbf{X}^T[\mathbf{y} - \boldsymbol{\pi}(\boldsymbol{\beta})], \end{aligned} \tag{7.6}$$

where $\mathbf{y} = [y_1\ y_2\ \cdots\ y_n]^T$ and $\boldsymbol{\pi}(\boldsymbol{\beta}) = [\pi(\mathbf{x}_1)\ \pi(\mathbf{x}_2)\ \cdots\ \pi(\mathbf{x}_n)]^T$. A closed-form solution for $\boldsymbol{\beta}$ cannot be obtained from the system

$$\mathbf{0} = \mathbf{X}^T[\mathbf{y} - \boldsymbol{\pi}(\boldsymbol{\beta})]. \tag{7.7}$$

However, Newton's method may be to iteratively approximate the solution. The updating formula for Newton's method (as was developed previously) is

$$\underset{q\times 1}{\boldsymbol{\beta}_k} = \underset{q\times 1}{\boldsymbol{\beta}_{k-1}} + (\underset{q\times q}{\mathbf{X}^T\mathbf{W}_{k-1}\mathbf{X}})^{-1}\underset{q\times n}{\mathbf{X}^T}[\underset{n\times 1}{\mathbf{y}} - \underset{n\times 1}{\boldsymbol{\pi}(\boldsymbol{\beta}_{k-1})}], \tag{7.8}$$

where (with a slight change of notation to emphasize the dependency of $\boldsymbol{\pi}$ on $\boldsymbol{\beta}$),

$$\begin{aligned} \pi_i(\boldsymbol{\beta}_{k-1}) &= [1 + \exp(-\mathbf{x}_i\boldsymbol{\beta}_{k-1})]^{-1}, \\ \boldsymbol{\pi}(\boldsymbol{\beta}_{k-1}) &= [\pi_1(\boldsymbol{\beta}_{k-1})\ \ \pi_2(\boldsymbol{\beta}_{k-1})\ \ \cdots\ \ \pi_n(\boldsymbol{\beta}_{k-1})], \end{aligned} \tag{7.9}$$

and

$$\underset{n\times n}{\mathbf{W}_{k-1}} = \text{diag}\{\pi_i(\boldsymbol{\beta}_{k-1})[1 - \pi_i(\boldsymbol{\beta}_{k-1})]\}. \tag{7.10}$$

7.3 Predicting the target

Let's review the logistic regression approach to prediction. The prediction function is constructed from the training data set D in essence, by computing the maximum likelihood estimator $\widehat{\boldsymbol{\beta}}$.[1] The prediction function is applied to an unlabeled predictor vector $\mathbf{x}_0$ to estimate $\pi(\mathbf{x}_0) = \Pr(Y_0 = 1|\mathbf{x}_0)$. Specifically, the logistic regression model specifies that

$$\widehat{\pi}(\mathbf{x}_0) = \left[1 + \exp(-\mathbf{x}_0\widehat{\boldsymbol{\beta}}\right]^{-1}. \tag{7.11}$$

The last step of the prediction process determines the prediction $\widehat{y}_0$ by comparing the estimated probability $\widehat{\pi}(\mathbf{x}_0) = \widehat{\Pr}(Y_0 = 1|\mathbf{x}_0)$ to a threshold p (usually .5). Formally, the comparison is made in the guise of decision rule g:

$$\widehat{y}_0 = g[p, \widehat{\pi}(\mathbf{x}_0)] = \begin{cases} 1, & \text{if } p \le \widehat{\pi}(\mathbf{x}_0), \\ 0, & \text{otherwise.} \end{cases} \tag{7.12}$$

The terms a *decision rule* and *classification function* are both used to describe equation (7.12).

[1] When y_0 is unknown and $\mathbf{x}_0$ is observed, then $\mathbf{z}_0$ presumed to be the outcome of the same process that generated the training set D. This statement includes sampling $\mathbf{z}_0$ from a population in which case the population from which $\mathbf{z}_0$ is drawn is presumed to be the same as the population from which D was drawn.

The choice of the threshold depends on the cost of incorrectly predicting the class membership of $\mathbf{z}_0$. Two errors are possible: predicting $\mathbf{z}_0 \in C_0$ when, in truth, $\mathbf{z}_0 \in C_1$, and predicting $\mathbf{z}_0 \in C_1$ when, in truth, $\mathbf{z}_0 \in C_0$. The costs associated with these errors may differ. For example, consider testing whether or not a bag containing an explosive device in an airport screening procedure. Incorrectly identifying a bag as containing an explosive device when it does not contain explosives is far less costly than the error of failing to identify a bag as containing an explosive device when it does contain explosives. If $y_0 = 1$ identifies the bag as containing a device, then it's best to set p substantially larger than .5 so that even weak evidence of a device (say, $\widehat{\pi}(\mathbf{x}_0) = .1$) will lead to an action that ameliorates the apparent danger.

7.3.1 Sensitivity and specificity

The choice of threshold, and hence, the definition of the prediction function, are addressed mathematically by investigate the sensitivity and specificity of the prediction function. Sensitivity and specificity are important in a number of situations involving binary targets.[2] They're commonly used in testing for a disease or condition. The presence of the condition in a individual (or unit) is identified by the target value $y = 1$. Hence, $y = 0$ corresponds to the absence of the condition in an observational unit $\mathbf{z}_0 = (y_0, \mathbf{x}_0)$. *Sensitivity* is the conditional probability estimate

$$\widehat{\Pr}\{g[p, \widehat{\pi}(\mathbf{x}_0)]) = 1 | y_0 = 1\} = \frac{\widehat{\Pr}\{g[p, \widehat{\pi}(\mathbf{x}_0)] = 1 \text{ and } y_0 = 1\}}{\widehat{\Pr}(y_0 = 1)}. \qquad (7.13)$$

Sensitivity of the prediction function is also referred to as the *true positive rate*[**?**]. In conventional testing situations, it's also desirable that the *specificity*, or estimated true negative rate $\widehat{\Pr}[g(\mathbf{x}, p) = 0 | y = 0]$ is large.

Computing estimates of sensitivity and specificity begins by evaluating the prediction function on a set of observations for which the targets are known. It's usually not advisable, but let's suppose that the $\widehat{\boldsymbol{\beta}}$ has been computed from the training set D, and then the prediction function is applied to all of the predictor vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$. The outcome is a set of actual and predicted label pairs, say $\{(y_1, g(\mathbf{x}_1, p)), \ldots, (y_N, g(\mathbf{x}_N, p))\}$. Let's suppose that the set has been reduced to a confusion matrix of the form shown in Table 7.1.[3] The table entries are the numbers of units actually belonging to C_i and predicted to belong to C_j, for $i, j = 0, 1$.

The sensitivity of the prediction function is the conditional probability estimate

[2] Sensitivity and specificity do not extend easily to more than two classes.

[3] Confusion matrices are discussed at length in Section 7.4.

Table 7.1 A confusion matrix summarizing the application of a prediction function to a test set consisting of n_{++} pairs of actual and predicted class labels. Assume that a target value $y_0 = 0$ identifies membership in C_0 whereas $y_0 = 1$ identifies membership in C_1.

	Predicted class		
Actual class	C_0	C_1	Total
C_0	n_{00}	n_{01}	n_{0+}
C_1	n_{10}	n_{11}	n_{1+}
Total	n_{+0}	n_{+1}	n_{++}

$$\widehat{\Pr}[g(\mathbf{x}, p) = 1 | y = 1] = \frac{n_{11}}{n_{1+}}, \tag{7.14}$$

where n_{1+} is the number of units actually belonging to class C_1, and n_{11} is the number of units actually belonging to class C_1 and also predicted to belong to C_1. Specificity is the estimated conditional probability of being predicted to belong to C_0 and given that the unit actually belongs to C_0:

$$\widehat{\Pr}[g(\mathbf{x}, p) = 0 | y = 0] = \frac{n_{00}}{n_{0+}}. \tag{7.15}$$

A third summary statistic is sometimes reported: the *false positive* rate

$$\widehat{\Pr}[g(\mathbf{x}, p) = 1 | y = 0] = \frac{n_{01}}{n_{0+}}. \tag{7.16}$$

The threshold p has an important role in determining sensitivity and specificity. Notice that if p is shifted from .5 to a smaller value, then more units will be predicted to belong to C_1; consequently, the estimated sensitivity n_{11}/n_{+1} (or estimated conditional probability of being predicted as a member of C_1 when actually a member of C_1) will tend to become larger. Conversely, the estimated specificity n_{00}/n_{+0} will be reduced as units shift from being predicted to belong to C_0 to C_1.

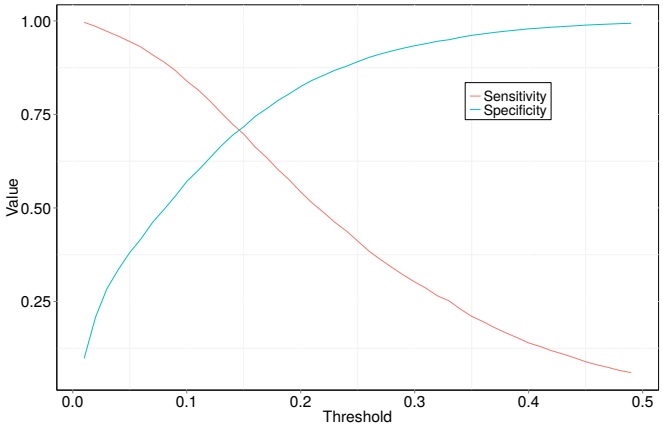


Fig. 1 Sensitivity and specificity plotted against the threshold p for a simple prediction function.

To illustrate what may happen with some simple prediction functions, we've graphed estimated sensitivity and specificity for a simple prediction function in Figure 1. The prediction function yields values of $\widehat{\Pr}(y_0 = 1|\mathbf{x}_0)$ that are all smaller than .5 for the test set, and so using $p = .5$ as the threshold produces no predictions of membership in C_1 for test set units. Hence, estimated sensitivity is 0 and estimated specificity is 1. The prediction function apparently has no predictive value because of the imbalance between specificity and sensitivity. On the other hand, if we hold sensitivity and specificity equally important, then the choice of $p = .15$ will result in both estimated sensitivity and specificity roughly equal and .72 in value since .72 is the approximate threshold at which the two curves intersect (Figure 1).

7.3.2 Accuracy rates for more than two classes

Though logistic regression involves only two classes, it's straightforward to extend the ideas of the last section to $g > 2$ classes or groups. The standard parameters used to describe the accuracy of a prediction function are the unconditional accuracy rate and the conditional accuracy rates.[4] The overall accuracy of a prediction function $g(\cdot|D)$ is the proportion of vectors $\mathbf{x}$ in a population, or vectors generated by a process, that are correctly labeled by $f(\cdot|D)$. This probability is unconditional since it doesn't depend on the class membership y or the prediction of the class membership $\widehat{y}$. We express the unconditional accuracy rate as

$$\gamma = \Pr[f(\mathbf{x}_0) = y_0|D] = \Pr(\widehat{y}_0 = y_0|D),$$

where $\mathbf{z}_0 = (y_0, \mathbf{x}_0)$ is a randomly selected observation from the population of interest. The target value is extended by defining it's value to be i if $\mathbf{z} \in C_i, i = 0, 1, \ldots, g-1$. As before, every $\mathbf{z}$ belongs to exactly one of the classes $C_0, C_1, \ldots, C_{g-1}$. The definition is remains useful if the random sampling condition is replaced with the condition that the observation in question is a realization of a process much like the process that generated the data set D.[5]

The accuracy rate for class $C_i, i = 0, 1, \ldots, g-1$, is the conditional probability that an observation belonging to C_i is predicted to belong to C_i. We express the conditional probability as

$$\alpha_i = \Pr(y_0 = \widehat{y}_0|y_0 = i).$$

[4] Sensitivity and specificity are both conditional accuracy rates.

[5] It has been implicitly assumed that accuracy assessment is carried out using the data set D from which the prediction rule has been constructed.

In the case of two classes, α_i represents the sensitivity (given $i = 1$) or the specificity (given $i = 0$). The conditional probability that a prediction of membership in class j is correct is

$$\varphi_j = \Pr(y_0 = \widehat{y}_0 | \widehat{y}_0 = j).$$

In this case, we're conditioning on the outcome $\widehat{y}_0 = j$. This probability is of interest after computing the prediction $\widehat{y}_0 = j$ and we question how likely the prediction is to be correct. For example, a drug test is reported to be positive—should the test result be accepted without dispute or is it possible that the test is incorrect? The question can be addressed by examining φ_j.

The aforementioned probabilities must be estimated in almost all situations. Superficially, the accuracy estimators are simple. We suppose for now a collection of actual and predicted labels has been created by applying the prediction function to a test set E of observation pairs.[6] The set is

$$L = [(y_0, \widehat{y}_0) | (y_0, \mathbf{x}_0) \in E]. \tag{7.17}$$

From these data, we may compute estimates of the accuracy rates γ, α_i, and φ_j. The estimates are easily computed from the confusion matrix shown in Table 7.2.

Table 7.2 A confusion matrix summarizing the application of a prediction function to a test set consisting of n_{++} pairs of actual and predicted class labels. The number of observations belonging to class i and assigned to class j is n_{ij}.

	Predicted class				
Actual class	C_0	C_1	$\cdots$	C_{g-1}	Total
C_0	n_{00}	n_{01}	$\cdots$	$n_{0,g-1}$	n_{0+}
C_1	n_{10}	n_{11}	$\cdots$	$n_{1,g-1}$	n_{1+}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	
C_{g-1}	$n_{g-1,0}$	$n_{g-1,1}$	$\cdots$	$n_{g-1,g-1}$	$n_{g-1,+}$
Total	n_{+0}	n_{+1}	$\cdots$	$n_{+,g-1}$	n_{++}

The estimator of the overall accuracy rate γ is the proportion of correctly predicted test set observations; i.e.,

$$\widehat{\gamma} = \frac{\sum_{j=0}^{g-1} n_{jj}}{n_{++}}.$$

The group-specific accuracy rate for group i, α_i, is the probability that an observation *belonging* to group i is correctly classified. The estimator of α_i is the proportion of test observations belonging to group i that are predicted

[6] The next section discusses *cross-validation*, a standard method of creating L and at the same time, insuring (in most situations) that the accuracy estimates are representative, i.e., nearly unbiased.

to belong to group i:

$$\widehat{\alpha}_i = \widehat{\Pr}(y_0 = \widehat{y} | y_0 = i) = \frac{n_{ii}}{n_{i+}}.$$

Similarly, the estimated probability that an observation *predicted* to belong to group j truly is a member of group j is

$$\widehat{\varphi}_j = \widehat{\Pr}(y_0 = \widehat{y} | \widehat{y} = j) = \frac{n_{jj}}{n_{+j}}.$$

7.4 Cross-validation

Let's take a closer look at accuracy assessment. As usual, let $D = \{(y_1, \mathbf{x}_1), \ldots, (y_n, \mathbf{x}_n)\}$ denote the data set. The usual approach to accuracy assessment begins by creating a test set E and training set R. Suppose that $R \cap E \neq \emptyset$. This situation presents a potentially serious problem—some observations (those in $R \cap E$) are being used for both building the prediction function and estimating the error of the prediction function. The resulting accuracy estimates often are optimistically biased.[7] The extent of bias is difficult to determine without an involved analysis.

There is a somewhat unique case in which the consequence of using the same observations in both sets can be seen with no trouble. Consider the 1-nearest neighbor prediction function, and suppose that $\mathbf{x}_0 \in R \cap E$. Suppose that there are no ties whatsoever and we investigate the accuracy of $f(\mathbf{x}_0|R)$ using the test observations in E. Since $\mathbf{x}_0 \in R$, $\mathbf{x}_0$ will be closest to itself and the nearest neighbor label will be its own label. The prediction of y_0 is certain to be correct, and every observation in $R \cap E$ will be correctly predicted. We learn nothing of how the function will perform when $\mathbf{x}_0 \notin R$ from these test observations. Even worse, the accuracy estimates are misleading (specifically, optimistically biased) because the prediction function performs better on the test data than a completely unrelated or independent data set. This phenomenon is known as *over-fitting*.

Over-fitting can be eliminated by insuring that no observation used to build the prediction function, i.e., $\mathbf{z} \in R$, is used as a test observation. This can be done using a random mechanism that divides D into subsets E and R such that $D = R \cup E$ and $R \cap E = \emptyset$. We'll approximate the prediction function $f(\cdot|D)$ that we intend to use by the function $f(\cdot|R)$ and apply $f(\cdot|R)$ to every test observation in E. The result is a set of predictions that are free of over-fitting bias. This process is called *validation* accuracy assessment.

Occasionally, D is too small to use for validation accuracy assessment because removing a sufficiently large test set results in $f(\cdot|R)$ being a poor

[7] Accuracy estimates are called *apparent* or *plug-in* estimates in the worst case of $D = R = E$.

approximation of $f(\cdot|D)$. Then, the accuracy of $f(\cdot|R)$ will be significantly less than the accuracy of $f(\cdot|D)$. Furthermore, when the sample is small, there will be a substantial degree of variation among accuracy estimates computed from different test sets. It would be wise not to trust the results derived from a single partition $\{E, R\}$ of D. The k-fold cross-validation algorithm is a solution to these accuracy estimation problems.

A cross-validation algorithm withdraws a set of observation pairs E from the training sample D to serve as a test set. The remaining observations in R are used to construct a prediction function $f(\cdot|R)$. Then $f(\cdot|R)$ is applied to each test set vector $\mathbf{x}_0 \in E$ to produce a prediction. The result is a set $L = \{(y_0, f[\mathbf{x}_0|R])|(y_0, \mathbf{x}_0) \in E\}$ from which a confusion matrix and estimates of the accuracy parameters γ, α_i, and φ_j for $i, j \in 0, 1 \ldots, g-1$ can be computed. Since the held-out sample was not used in constructing the prediction function, over-fitting bias has been eliminated.

So far, nothing is new. What is new with cross-validation is that not one test set is drawn from D, but instead k, where $1 < k \leq n$. The hold-out test sets are $E_1, E_2, \ldots, E_k$, and the training sets are $R_i = E_i^c$, for $i = 1, 2, \ldots, k$. The outcomes of predicting $\mathbf{x}_0 \in E_i$ are collected in the set $L_i = \{(y_0, f[\mathbf{x}_0|R_i])|(y_0, \mathbf{x}_0) \in E_i\}$. After all test sets have been processed, the confusion matrix is computed from $L = \cup_{i=1}^{k} L_i$. In this scenario, the training sets R_i can be made to be closer to D in size, with the usual effect that the prediction functions $f(\cdot|R_i)$ are good approximations of $f(\cdot|D)$. Finally, the accuracy estimates will be relatively precise because L is large (provided that D is representative of the population or process to which the prediction function will be applied).

It's efficient if $E_1, \ldots, E_k$ form a partition of D so that every observation is a target exactly once.[8] A *partition* of D is a set of sets $\{D_1, D_2, \ldots, D_r\}$ with the properties that $D = \cup D_i$ and $D_i \cap D_j = \emptyset$, for each pair $(D_i, D_j), i \neq j$. Formally, a *k-fold* cross-validation algorithm partitions D as k disjoint subsets each containing approximately n/k observations. There are a variety of ways to implement the algorithm. For example, each observation $i = 1, 2, \ldots, n$, may be assigned a random number drawn from $\{1, 2, \ldots, k\}$ that identifies membership in one of the subsets $E_1, \ldots, E_k$. The cross-validation algorithm iterates over $i \in \{1, 2, \ldots, k\}$. On the ith iteration, E_i is the test set and $R_i = E_i^c$ is the training set. The prediction function $f(\cdot|R_i)$ is constructed and used to predict y_0 for each $\mathbf{x}_0 \in E_i$. The results are collected in L_i. A confusion matrix is built and accuracy estimates are computed from $L = \cup_i L_i$ at the completion of the k iterations. A commonly used value for k is 10. Another popular choice is $k = n$, also known as hold-one-out cross-validation.

As the cross-validation accuracy estimates will depend on the initial partitioning of D, some number of repetitions of the algorithm may be carried out using different random partitions. Sufficiently many repetitions will reduce

[8] A lazy alternative is to draw independent random samples $E_1, \ldots, E_k$. A test set may appear in more than one test set. This presents no risk of bias, but there's less information gained on the second and subsequent predictions of a test observation.

the variation associated with random partitioning to a negligible fraction of the estimate. At completion, the estimates from different repetitions are aggregated in the construction of the confusion matrix.

7.5 Exercises

1. Starting from equation (7.1), show

   a. $\pi(\mathbf{x}_i) = [1 + \exp(-\mathbf{x}_i\boldsymbol{\beta})]^{-1} = \dfrac{\exp(\mathbf{x}_i\boldsymbol{\beta})}{[1 + \exp(\mathbf{x}_i\boldsymbol{\beta})]}$,
   b. $1 - \pi(\mathbf{x}_i) = [1 + \exp(\mathbf{x}_i\boldsymbol{\beta})]^{-1}$,
   c. $\log[1 - \pi(\mathbf{x}_i)] = -\log\{[1 + \exp(\mathbf{x}_i\boldsymbol{\beta})]\}$,
   d.
   $$\underset{q\times 1}{\frac{\partial \log[1 - \pi(\mathbf{x}_i)]}{\partial \boldsymbol{\beta}}} = -\pi(\mathbf{x}_i)\mathbf{x}_i^T \tag{7.18}$$

   (Requires multivariable calculus).

2. This exercise guides the reader through an analysis of sensitivity and specificity for a problem in which the number of observations in two classes are wildly imbalanced. These data are measurements on light intensity received from approximately 5000 stars. The vast majority of stars are presumed not to have any planets (obliquely referred to as *exoplanets*). Those stars with exoplanets exhibit varying degrees of regularly-occurring attenuation in the light intensity received by the Kepler telescope. The data come from a Kaggle competition in which objective was to build a prediction function from the data that will correctly classify stars as either having at least one exoplanet or not.[9]
   So that you may concentrate on the analysis of sensitivity and specificity, I have provided a `Python` script that reduces the data to a matrix of predictor vectors and a vector of targets (`exoplanet.py`). The objective is to estimate sensitivity and specificity over a range of thresholds used in the decision rule (equation 7.12), and plot the estimates against threshold to produce a figure similar to Figure 1.
   Exercise 3 also uses these data and computes sensitivity and specificity estimates using these data (except that it's done using cross-validation to avoid over-fitting). It's advisable, but not necessary to create functions so that the code is easily re-used for exercise 3.

   a. Read the data file and create $\mathbf{X}$ and $\mathbf{y}$ using the `Python` script `exoplanetsReducer.py`

[9] https://www.kaggle.com/keplersmachines/kepler-labelled-time-series-data.

b. Compute the maximum likelihood estimator $\widehat{\boldsymbol{\beta}}$ via logistic regression. Use your direct search or Newton's method algorithm, or use the `Python`) implementation[10].
c. Create two lists: the first containing the actual labels[11] and second containing the estimated probabilities $\widehat{\boldsymbol{\pi}}(\mathbf{x}_1), \ldots, \widehat{\boldsymbol{\pi}}(\mathbf{x}_n)$.
d. Create a list of thresholds, initialize a confusion matrix, and initialize lists to contain the sensitivity and specificity estimates, for instance,

```
thresholds = np.arange(.001,.1,.001)
cm = np.zeros(shape = (2, 2))
sensitivity = [0]*len(thresholds)
```

e. Iterate over the list of thresholds and with each threshold in the list,
   i. evaluate a prediction function g (equation 7.12) using the threshold and the estimated probabilities of membership. The result ought to be a list of group membership predictions.
   ii. Compute estimates of sensitivity and specificity using the predictions of group membership and the actual group memberships. Save the estimates in sensitivity and specificity lists created in the previous step.
f. Plot the estimates of sensitivity and specificity against threshold. Save the plot.
g. Suppose that the objective is to construct a prediction function with an estimated sensitivity of .8 or greater. Given that constraint, what threshold maximizes *specificity*, and what is the estimated specificity of this best prediction function?

3. This exercise guides the reader through the construction of a cross-validation algorithm. Use the exoplanet data described in exercise 2. At completion of the cross-validation algorithm, re-create the plot of estimated sensitivity and specificity. The algorithm described below begins by initializing empty lists to save the probability estimates and actual labels. As the program iterates over each of the $k = 10$ folds, the lists are extended with the kth fold *test set or holdout* probability estimates and actual labels. For example, at the completion of the first iteration, the lists will contain $\pi(\mathbf{x}_1), \pi(\mathbf{x}_2), \ldots, \pi(\mathbf{x}_s)$ and $y_1, y_2, \ldots, y_s$—the hold-out probability estimates and labels for the first set of test observations.[12] At the completion of the second iteration, the lists are extended by concatenating the probability estimates and actual labels obtained from the second set of test observations.

[10] See `http://scikit-learn.org`.

[11] Say, `actual = [int(y[0]) for y in np.matrix.tolist(Y)]`.

[12] I've ignored the original ordering of the observations and assigned subscripts to correspond to the test set.

a. Import the `Python` module `random` and generate a list of n random integers (n is the number of observations in the data set) drawn randomly and with replacement from $\{0, 1, \ldots, k-1\}$. The instruction `sampleID = [random.choice(range(k)) for i in range(n)]` will create the randomized list of integers. This list provides the assignments of each observation to one of the k test sets. The test sets will have approximately the same number of observations ($\sim n/k$).
b. Construct a `for loop` to iterate over the k folds. On each iteration, determine the indexes of the training and test sets. For example, if the iteration (or fold) counter is `j`, then the indexes of the test set observations are contained in the list

```
sIndex = [i for i in range(n) if sampleID[i] == j]
```

c. Create a matrix consisting of the jth fold training set predictors. You can create the training set predictor matrix and target vector by slicing, e.g., the jth fold training set target vector is `D.Y[rIndex]`, where `rIndex` is the list of observations that are to be used for training on the jth fold of the cross-validation algorithm. The training set predictor matrix can be extracted by slicing in a similar fashion.
d. Compute the coefficient vector $\widehat{\boldsymbol{\beta}}$ using the jth fold training predictor matrix and target vector.
e. Using the coefficient vector and the test set predictor matrix, compute probability estimates for the test set observations. Store the results by extending the list of probability estimates and actual values. Note that both the `.extend` and `.append` operators *update* a list and cannot be used to create a new variable. Therefore, extending the list of probability estimates is carried out with an instruction of the form

```
piHat.extend([1/(1 + np.exp(-x*b)) for x in D.X[sIndex,:]])
```

f. Also extend the list of actual values so that the probability estimates and actual values are correctly aligned.[13]
g. At the completion of k folds, compute estimates of sensitivity and specificity using the predictions of group membership and the actual group memberships.
h. Plot the estimates of sensitivity and specificity against threshold.
i. Provide your annotated code along with the plot.

[13] Aligned in this context means that the ith position in the probability estimate list has been computed from $\mathbf{x}_i$, say, and that the ith position in the list of actual values is occupied by y_i.

4. Complete the tutorial beginning on page 329 of *Algorithms for Data Science*. This tutorial depends on the output of the preceding tutorial (beginning on page 319). I've posted the code for preceding tutorial on Moodle. The tutorial instructs the reader to download the Federalists Papers as a text file from `gutenber.org`. There's a critical missing author name (The first tutorial instructs the reader to fix the file). The file posted on Moodle has been fixed, so it's easier to use it.

Chapter 8
Forecasting

Let us begin with an important subject contained within the broader subject of predictive analytics: forecasting. The aim of forecasting is to predict the outcome of a process at some time in the future. Whatever information is available to us at the present time may be used, and we need a mathematical function that will transform the relevant information to a prediction. An example is an attempt forecast future stock prices using observations on the end-of-the-day, or closing prices for a particular stock, or perhaps a set of s stocks.

The *target* is the closing price of a particular stock. Let t denote the current time step, and $t + \tau$ denote the time step for which the prediction is desired. The definition of time step is up to the analyst, for example a time step may be a day or a minute. In any case, x_t denotes the stock price at time step t. Since the prediction is to be computed for τ time steps in the future, the *target* is $x_{t+\tau}$. To emphasize the distinction between the observed predictor variables and the unobserved (future) target variables, we'll use the notation

$$y_t = x_{t+\tau} \tag{8.1}$$

to denote the target of the prediction effort. I've purposely changed the subscript on y to t to denote that the forecast is computed at time step t using a set of values observed up to and including time step t.

A simple approach to the forecasting problem sets the predictor x_t to be the observed value of the target at time step t. For example, time steps may correspond to days and x_t may be the closing price of a stock (the asking price of the stock at the end of the trading day).[1]

A *prediction (or forecasting) function* f will be constructed from a training set. The training set is a set of pairs consisting of a predictor vector $\mathbf{x}_t$ and an observed target (either a scalar, or real number, y_t), or a vector $\mathbf{y}_t$. For

[1] It's common that the time step is shorter—hours, minutes, even seconds in high-volume algorithmic trading.

now, suppose that the target is a scalar y_t as defined in equation (8.1). The input to the prediction function is a *predictor vector* denoted by $\mathbf{x}_t$.

We suppose that the predictor vector $\mathbf{x}_t$ consists p past values observed on the stock and collected as a row vector

$$\underset{1\times p}{\mathbf{x}_t} = [x_{t-p+1} \quad x_{t-p+2} \quad \cdots \quad x_t].$$

The prediction function might be a linear predictor in which case it produces the prediction by combining the target values as a sum, say,

$$f(\mathbf{x}_t|D) = \sum_{i=1}^{p} x_{t-p+i} h_i. \tag{8.2}$$

Most of the time, the prediction of y_t will be denoted as $\widehat{y}_t$. In statistics, the hat signifies that the symbol under the hat is a the target of estimation or prediction (prediction includes forecasting). The linear predictor (equation 8.2) looks like a fitted regression model though the standard statistical notation would denote the coefficients as $\widehat{\beta}_0, \ldots, \widehat{\beta}_{p-1}$ rather than $h_1, \ldots, h_p$.[2] To simplify notation, the coefficients $h_1, \ldots, h_p$ are collected as a p-length column vector $\mathbf{h}$ so that forecast of $y_{t+\tau}$ can be written as the inner or dot product between $\mathbf{x}_t$ and $\mathbf{h}$:

$$\widehat{y}_t = \underset{1\times p}{\mathbf{x}_t} \underset{p\times 1}{\mathbf{h}}.$$

The prediction function is written as $f(\cdot|D)$ to express the reliance on the data set in determining the coefficient values. In essence, the prediction function consists of three things: a rule—compute the inner product of the coefficients and the predictor vector, a vector of coefficients $\mathbf{h}$ determined from a training set, and a target vector ($\mathbf{x}_t$). Before expanding on the subject of prediction functions and how to train them on a data set, let's expand the notation to accommodate multivariate targets (in contrast to the scalar target discussed above).

For example, it's helpful to collect the predictor vectors as one matrix and the targets either as a vector in the case that the targets are scalars, or as a matrix in the case that the targets are also vectors. Suppose that the targets are vectors of length s corresponding to s different stocks and forecasting is for τ time steps in the future. Let's suppose that there are $q = p \times s$ predictor variables. Then, the predictor matrix $\mathbf{X}$ is constructed by stacking the predictor vectors as $N - \tau$ rows:

[2] Furthermore, $\beta_0, \ldots, \beta_{p-1}$ are parameters that describe the expected value of a random variable Y_t.

$$\underset{(N-\tau)\times q}{\mathbf{X}} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{N-\tau} \end{bmatrix} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,q} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,q} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N-\tau,1} & x_{N-\tau,2} & \cdots & x_{N-\tau,q} \end{bmatrix}.$$

The target matrix is

$$\underset{(N-\tau)\times s}{\mathbf{Y}} = \begin{bmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_{N-\tau} \end{bmatrix} = \begin{bmatrix} y_{1,1} & \cdots & y_{1,s} \\ y_{2,1} & \cdots & y_{2,s} \\ \vdots & \ddots & \vdots \\ y_{N-\tau,1} & \cdots & y_{N-\tau,s} \end{bmatrix}.$$

If the targets were scalars rather than vectors, then $\mathbf{y}^T = [y_1 \ \cdots \ y_{N-\tau}]$ where $\mathbf{y}^T$ is the transpose of $\mathbf{y}$.

Task: Suppose that the aim is to make two forecasts for two different future values, say $y_{i,t+\tau}$ and $y_{i,t+\gamma}$ for each stock, $i = 1, 2, \ldots, s$. What would the target matrix look like? In other words, write the terms $(\ldots, x_{t-1}, x_t, x_{t+1})$ that form the target vector at time t. What are its dimensions?

8.1 Example

In many situations in which data are observed in a chronological sequence, consecutive observations tend to be alike, that is, *serially correlated*. If this is the case, then past observations contain predictive information about near-term future observations. We may exploit this information be constructing predictor vectors from the recent past values. For example, suppose that the current, and previous two observations are used to predict a target one time step ahead of the current value. The data would be similar to those shown in Table 8.1. Note that the target value at time step t is incorporated into the predictor at time $t + 1$.

8.2 Multivariate targets

Our interest is in simultaneously forecasting $s \geq 1$ stocks in an analogous manner. An observation on stock i obtained at time t is identified by subscripts, and so is denoted by $x_{t,i}$. Similarly, the target value for stock i is denoted as $y_{t,i} = x_{t+\tau,i}$. Now, the targets are collected in the row vector

$$\mathbf{y}_t = [x_{t+\tau,1} \ \ x_{t+\tau,2} \ \cdots \ x_{t+\tau,s}].$$

Table 8.1 Example data constructed from a time series of observations on a target stock. The target is the closing price one time step ahead of the current value (the closing price on day t).

t	$\mathbf{x}_t$	$y_t = x_{t+1}$
3	[1.3, 1.8, 1.7]	1.9
4	[1.8, 1.7, 1.9]	2.4
5	[1.7, 1.9, 2.4]	2.5
⋮	⋮	⋮

Task: write out the values composing the observed series of observations $(x_1, x_2, \ldots, x_6)$.

If data collection ends at day N, then we would observe $\mathbf{x}_N$ but not $y_N = x_{N+1}$. The data set would be used to construct a prediction function that could be used $\mathbf{x}_N$ to forecast y_N.

The vector $\mathbf{y}_t$ is called the target vector, or simply, the target.

We may as well use the past information encapsulated in the current and past p time steps in all s stocks to forecast $\mathbf{y}_t$. Then, the predictor vector becomes

$$\underset{1\times q}{\mathbf{x}_t} = [\underbrace{x_{t-p+1,1}\ x_{t-p+2,1}\ \cdots\ x_{t,1}}_{\text{stock 1}}\ \cdots\ \underbrace{x_{t-p+1,s},\ x_{t-p+2,s},\ \cdots\ x_{t,s}}_{\text{stock } s}],$$

where its length is $q = p \times s$.

Let us suppose that the data set is constructed from observations collected up to and including time step N. The data set is then

$$D = \{(\mathbf{y}_1, \mathbf{x}_1), \ldots, (\mathbf{y}_{N-\tau}, \mathbf{x}_{N-\tau})\}$$

because $\mathbf{y}_{N-\tau} = [x_{N,1}\ \cdots\ x_{N,s}]$ and we have no observations beyond the current time step (N).