CLASS NOTES

MATH 437/537
Multivariate Analysis

SPRING 2020

Contents

1 Linear Regression **1**

1.1 Basic Ideas 1

1.2 Prediction 4

2 Subset Selection Methods **8**

2.1 Best Subset Selection 8

2.2 Stepwise Selection 9

2.3 Formulas for Choosing a Single Model 9

2.4 An Example 11

3 Cross-Validation **16**

3.1 Training Sets and Test Sets 16

3.2 Test Sets Aren't As Good As Actual New Data 16

3.3 Leave-One-Out-Cross-Validation 17

3.4 k-fold Cross-validation 19

4 Shrinkage Methods **22**

4.1 Ridge Regression 22

4.2 The Lasso 30

4.3 The Elastic Net 34

5 Vectors and Matrices **37**

5.1 Vectors 37

5.2 Basic Facts About Matrices 40
5.3 Determinants 42
5.4 Singular and Nonsingular Matrices 44
5.5 Orthogonal and Orthonormal Matrices 47
5.6 Eigenvectors and Eigenvalues 47
5.7 Symmetric Matrices 51
5.8 Quadratic Forms 55
5.9 Singular Value Decomposition 56
5.10 Another Look at Linear Regression 58
5.11 Another Look at Ridge Regression 61

6 Principal Components **63**

6.1 Introduction 63
6.2 Principal Components 64
6.3 Principal Components Based on Standardized Variables 68
6.4 Principal Components Regression (PCR) 75
6.5 Using Cross-Validation to Choose the Number of Principal Components . . . 83
6.6 Comparing Principal Components Regression to Linear Regression and Ridge Regression 84

7 Classification **86**

7.1 Random Vectors 86
7.2 Introduction to Classification 89
7.3 Bayes Classifiers 89

7.4 Linear Discriminant Analysis 93
7.5 Estimating the Misclassification Rate 98
7.6 Quadratic Discriminant Analysis 108
7.7 Logistic Regression 112
7.8 k Nearest Neighbors Classification 121

8 Support Vector Machines **128**

8.1 The Maximal Margin Classifier 128
8.2 Lagrange Multipliers for Inequality Constraints 133
8.3 Computing the Maximal Margin Classifier 136
8.4 Non-Separable Data Sets 138
8.5 Kernels 141

9 Clustering **150**

9.1 Hierarchical Clustering 151
9.2 K-Means Clustering 163

1 Linear Regression

1.1 Basic Ideas

In the standard linear model we observe a random vector $\mathbf{Y} = (y_1, ..., y_n)$. Then for each $i = 1, ..., n$ we observe predictor variables $x_{i1}, ..., x_{ip}$. The model is

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i \text{ for } i = 1, ..., n$$

where $\beta_0, ..., \beta_p$ are parameters and ε_i is an unobserved random variable with mean 0. In this discussion we will further assume that $\varepsilon_1, ..., \varepsilon_n$ are i.i.d. with unknown variance σ^2.

Now let $\mathbf{X}$ be the $n \times (p+1)$ matrix whose first column is a column of 1s, and whose ij element is x_{ij}, let $\boldsymbol{\beta} = (\beta_0, \beta_1, ..., \beta_p)^T$, and let $\boldsymbol{\varepsilon} = (\varepsilon_1, ..., \varepsilon_n)^T$. We can write the model in matrix form as

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

Since $E(\boldsymbol{\varepsilon}) = \mathbf{0}$, $E(\mathbf{Y}) = \mathbf{X}\boldsymbol{\beta}$. Because we are assuming that $\varepsilon_1, ..., \varepsilon_n$ are independent with common variance σ^2, $y_1, ..., y_n$ are independent, with variance $V(y_i) = \sigma^2$.

We want to estimate $\boldsymbol{\beta}$. The least-squares estimate is the vector $\hat{\boldsymbol{\beta}}$ that minimizes the sum of squares

$$S = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i1} - \cdots - \beta_p x_{ip})^2$$

We can find the minimum by setting the partial derivatives of the sum of squares equal to 0. Let $x_{i0} = 1$ for $i = 1, ..., n$. We can then write the sum of squares as

$$S = \sum_{i=1}^{n} (y_i - \sum_{j=0}^{p} \beta_j x_{ij})^2$$

For $i = 1, ..., n$, let $\mu_i = \sum_{j=0}^{p} \beta_j x_{ij}$. Then

$$S = \sum_{i=1}^{n} (y_i - \mu_i)^2$$

We now compute the partial derivatives of the sum of squares with respect to β_j. Note that $\dfrac{\partial \mu_i}{\partial \beta_j} = x_{ij}$.

For each $j = 0, ..., p$,

$$\frac{\partial S}{\partial \beta_j} = \sum_{i=1}^{n} \frac{\partial}{\partial \beta_j} (y_i - \mu_i)^2 = -2 \sum_{i=1}^{n} \frac{\partial \mu_i}{\partial \beta_j} (y_i - \mu_{ij}) = -2 \sum_{i=1}^{n} x_{ij} (y_i - \mu_{ij}) = 0$$

We compute the estimate of $\boldsymbol{\beta}$ by solving the system of equations

$$\sum_{i=1}^{n} x_{ij}(y_i - \mu_i) = 0 \quad \text{for } j = 1, ..., p$$

Let $\mathbf{x}_j$ denote the jth column of $\mathbf{X}$. Writing the system of equations above as a dot product yields

$$\mathbf{x}_j^T(\mathbf{Y} - \boldsymbol{\mu}) = 0 \quad \text{for } j = 1, ..., p$$

In matrix form, this is

$$\mathbf{X}^T(\mathbf{Y} - \boldsymbol{\mu}) = \mathbf{0}$$

or equivalently

$$\mathbf{X}^T\mathbf{Y} = \mathbf{X}^T\boldsymbol{\mu}$$

Now, because $\boldsymbol{\mu} = \mathbf{X}\boldsymbol{\beta}$, the system of equations is

$$\mathbf{X}^T\mathbf{Y} = \mathbf{X}^T\mathbf{X}\boldsymbol{\beta}$$

These are called the **normal equations**. Solving the normal equations yields the least-squares estimator

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

The fitted values are

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

In particular, the ith fitted value is

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_p x_{ip}$$

The residuals are

$$\mathbf{e} = \mathbf{Y} - \hat{\mathbf{Y}}$$

In particular, the ith residual is

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

Stochastic Properties of Least-Squares Estimators

The model is $\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$. where the errors $\varepsilon_1, ..., \varepsilon_n$ are i.i.d. with mean 0 and variance σ^2. In vector notation, the assumption is that $E(\boldsymbol{\varepsilon}) = \mathbf{0}$ and $\text{Cov}(\boldsymbol{\varepsilon}) = \sigma^2\mathbf{I}$. With this assumption, we can derive various stochastic properties of the least-squares estimators.

- $E(\mathbf{Y}) = \mathbf{X}\boldsymbol{\beta}$.

  This follows from the fact that $\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$, with $E(\boldsymbol{\varepsilon}) = \mathbf{0}$.
  Then $E(\mathbf{Y}) = E(\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}) = \mathbf{X}\boldsymbol{\beta} + E(\boldsymbol{\varepsilon}) = \mathbf{X}\boldsymbol{\beta}$.

- $E(\hat{\boldsymbol{\beta}}) = \boldsymbol{\beta}$.

  To see this, note that
  $E(\hat{\boldsymbol{\beta}}) = E[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}] = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T E(\mathbf{Y}) = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = \boldsymbol{\beta}$.

- $E(\hat{\mathbf{Y}}) = \mathbf{X}\boldsymbol{\beta}$.

  To see this, note that
  $E(\hat{\mathbf{Y}}) = E(\mathbf{X}\hat{\boldsymbol{\beta}}) = \mathbf{X}E(\hat{\boldsymbol{\beta}}) = \mathbf{X}\boldsymbol{\beta}$

- $E(\mathbf{e}) = \mathbf{0}$.

  This follows from the fact that $\mathbf{e} = \mathbf{Y} - \hat{\mathbf{Y}}$. Therefore
  $E(\mathbf{e}) = E(\mathbf{Y}) - E(\hat{\mathbf{Y}}) = \mathbf{X}\boldsymbol{\beta} - \mathbf{X}\boldsymbol{\beta} = \mathbf{0}$.

Estimation of $\boldsymbol{\sigma^2}$

The error variance, $V(\varepsilon_i)$, is denoted σ^2. Since $E(\varepsilon_i) = 0$, $\sigma^2 = E(\varepsilon_i^2)$. If we knew the values of $\varepsilon_1, ..., \varepsilon_n$, we would estimate σ^2 with the mean of the squares of the ε_i: $\frac{\sum_{i=1}^n \varepsilon_i^2}{n}$. Since we don't know the values of ε_i, we use the residuals e_i instead. It is the case that the residuals tend to be smaller than the errors. For this reason we divide the sum of the squared residuals by a number somewhat smaller than n. It can be shown that the appropriate divisor is $n - p - 1$. We therefore estimate σ^2 with

$$s^2 = \frac{\sum_{i=1}^n e_i^2}{n - p - 1} = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n - p - 1}$$

Measuring Goodness-of-Fit

The linear model fits well if the fitted values $\hat{\mathbf{Y}}$ are close to the observations $\mathbf{Y}$. The statistic most often used to measure goodness-of-fit is the coefficient of determination, denoted R^2.

$$R^2 = \frac{\sum_{i=1}^n (y_i - \bar{y})^2 - \sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2} = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

We can describe R^2 as follows: If we had to predict the y-values without knowing the values of the predictor variables x_{ij}, we would estimate each y-value with the average $\bar{y}$ of the y-values,

and the residuals would be $y_i - \bar{y}$. When we can use the predictor variables to estimate the y-values, the residuals will be $y_i - \hat{y}_i$. The estimates using the predictor variables will tend to be closer to the true y-values than those made without, so the sum of squared residuals $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ will be smaller than $\sum_{i=1}^{n}(y_i - \bar{y})^2$. R^2 is the reduction in the sum of squared residuals obtained by using the predictor variables to estimate the y-values, expressed as a proportion of the sum of squared residuals when the predictor variables are not used.

1.2 Prediction

Now assume we want to predict the value of a future observation, y_0, that is independent of the observed values $y_1, ..., y_n$. We will refer to y_0 as the *target*. We know that the values of the predictor variables associated with y_0 are $\mathbf{x}_0 = (1, x_{01}, ..., x_{0p})^T$, so that

$$y_0 = \beta_0 + \beta_1 x_{01} + \cdots + \beta_p x_{0p} + \varepsilon_0 = \mathbf{x}_0^T \boldsymbol{\beta} + \varepsilon_0$$

The predicted value for y_0 is

$$\hat{y}_0 = \hat{\beta}_0 + \hat{\beta}_1 x_{01} + \cdots + \hat{\beta}_p x_{0p} = \mathbf{x}_0^T \hat{\boldsymbol{\beta}}$$

Since $E(\hat{\boldsymbol{\beta}}) = \boldsymbol{\beta}$,

$$E(\hat{y}_0) = \mathbf{x}_0^T \boldsymbol{\beta}$$

The difference $\hat{y}_0 - y_0$ between the predicted value and the target is the *prediction error* in y_0. If $\hat{y}_0$ is a good predictor of y_0, the prediction error will tend to be small. The quantity most commonly used to assess the performance of a prediction is the *mean square error of prediction* (MSEP).

$$\text{MSEP} = E[(\hat{y}_0 - y_0)^2]$$

We will compute MSEP for the predictor $\hat{y}_0 = \mathbf{x}_0^T \hat{\boldsymbol{\beta}}$.

$$\begin{aligned}
\text{MSEP} &= E[(\hat{y}_0 - y_0)^2] \\
&= E[(\hat{y}_0 - \mathbf{x}_0^T \beta - \varepsilon_0)^2] \\
&= E[(\hat{y}_0 - E(\hat{y}_0) - \varepsilon_0)^2)] \\
&= E[(\hat{y}_0 - E(\hat{y}_0)^2] - 2E[\varepsilon_0(\hat{y}_0 - E(\hat{y}_0))] + E(\varepsilon_0^2)
\end{aligned}$$

We examine each of these terms:

- $E[(\hat{y}_0 - E(\hat{y}_0))^2] = V(\hat{y}_0)$, the variance of the predictor $V(\hat{y}_0)$.

- To evaluate $E[\varepsilon_0(\hat{y}_0 - E(\hat{y}_0))]$, note that $\hat{y}_0$ is a function of the observed values $\mathbf{Y} = (y_1, ..., y_n)$ and ε is independent of $y_1, ..., y_n$. Therefore $E[\varepsilon_0(\hat{y}_0 - E(\hat{y}_0))] = E(\varepsilon_0)E[\hat{y}_0 - E(\hat{y}_0)] = (0)(0) = 0$.
- $E(\varepsilon^2) = \sigma^2$ is the variance of the target y_0.

We can therefore write

$$\text{MSEP} = V(\hat{y}_0) + V(y_0)$$

Thus MSEP is the sum of the variance of the predictor and the variance of the value to be predicted. Because we assume the model is known, we can provide formulas to estimate these variances. $V(y_0) = \sigma^2$, and it can be shown that $V(\hat{y}_0) = \sigma^2 \mathbf{x}_0^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_0$. We estimate σ^2 with s^2 to obtain the estimator MSEP $\approx s^2(1 + \mathbf{x}_0^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_0)$.

So far we have assumed that the model $y_0 = \mathbf{x}_0^T\boldsymbol{\beta} + \varepsilon_0$ is correct. Now we discuss MSEP under the more realistic situation in which we don't know the correct model. Assume that $y_0 = \mu_0 + \varepsilon_0$, where $\mu = E(y_0)$ is an unknown constant. Let $\hat{y}_0$ be any estimator of y_0 (not necessarily from a linear model), and let $\mu_0^* = E(\hat{y}_0)$. The error in the estimate is $\hat{y}_0 - y_0$. There are three quantities that are useful in analyzing the error.

1. Bias: The bias is $E(\hat{y}_0 - y_0) = \mu_0^* - \mu_0$. The bias is the difference between the mean of the predictor and the mean of the value to be predicted. The bias is the average error.
2. Variance of the predictor: The variance of the predictor is $V(\hat{y}_0)$. A predictor with a large variance will give very different values from prediction to prediction. A predictor with a small variance will give similar values every time.
3. The irreducible variance: This is the variance of the quantity to be predicted, $V(y_0) = V(\varepsilon_0)$. There is no way to estimate ε_0, since it is random. The larger the value of $V(\varepsilon)$, the more difficult it is to predict y_0.

We can derive an expression for MSEP in terms of the bias, the variance of the predictor, and the irreducible variance, as follows.

$$\begin{aligned} \text{MSEP} = E[(\hat{y}_0 - y_0)^2] &= E[[\hat{y}_0 - \mu_0^* + \mu_0^* - \mu_0 - \varepsilon_0]^2] \\ &= E[[(\hat{y}_0 - \mu_0^*) + (\mu_0^* - \mu_0) - \varepsilon_0]^2] \end{aligned}$$

Note that $\mu_0^* - \mu_0$ is constant. We can therefore multiply out to obtain

$$\begin{aligned} E[(\hat{y}_0 - y_0)^2] &= E[(\hat{y}_0 - \mu_0^*)^2] + [E(\hat{y}_0 - \mu_0)]^2 + E(\varepsilon_0)^2 + 2(\mu_0^* - \mu)E(\hat{y}_0 - \mu_0^*) \\ &\quad -2E[\varepsilon_0(\hat{y}_0 - \mu_0^*)] - 2E(\varepsilon_0)(\mu_0^* - \mu_0) \end{aligned}$$

Now $E(\hat{y}_0 - \mu_0^*) = 0$, $E(\varepsilon) = 0$, and as explained earlier, $E[\varepsilon_0(\hat{y}_0 - \mu_0^*))] = 0$. Therefore

$$\text{MSEP} = E[(\hat{y}_0 - \mu_0^*)^2] + (\mu_0^* - \mu_0)^2 + E(\varepsilon_0)^2$$

We examine each of these terms.

- $E[(\hat{y}_0 - \mu_0^*)^2] = V(\hat{y}_0)$, the variance of the predictor.
- $(\mu_0^* - \mu_0)^2$ is the square of the bias of the predictor.
- $E(\varepsilon_0)^2$ is the irreducible variance.

Therefore we can write

$$\text{MSEP} = \text{Variance} + \text{Bias}^2 + \text{Irreducible Variance}$$

As mentioned above, the irreducible variance is the variance of the target and contributes to MSEP unless the target is constant. Ideally, then, we try to find predictors with small bias and small variance. This is generally difficult, as models with smaller variance tend to have larger bias, and vice versa. This phenomenon is known as the *bias-variance tradeoff*, which we now describe.

The Bias-Variance Tradeoff

Imagine that we have a vector of n observations $\mathbf{Y} = (y_1, ..., y_n)^T$ that follow a model $\mathbf{Y} = \boldsymbol{\mu} + \boldsymbol{\varepsilon}$, where $\boldsymbol{\mu}$ is a vector of constants, and $\boldsymbol{\varepsilon}$ is vector of random variables. We predict $\mathbf{Y}$ with a linear model $\hat{Y} = \mathbf{X}\hat{\boldsymbol{\beta}}$. If we include $n-1$ predictors, then $\mathbf{X}$ will have n columns (including an intercept), and there will be an exact solution $\hat{\mathbf{Y}} = \mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{Y}$ that fits the observed data perfectly. However, this is not likely to be a good model. The problem is that we want to predict new unobserved values of Y, not the ones we have already observed. Therefore we want our fitted values to be close to $\boldsymbol{\mu}$, not close to $\mathbf{Y} = \boldsymbol{\mu} + \boldsymbol{\varepsilon}$. In fact, a

predictor $\hat{\mathbf{Y}}$ that is very close to $\mathbf{Y}$ is undesirable, because it captures the random variation $\boldsymbol{\varepsilon}$ in addition to the mean $\boldsymbol{\mu}$. Such models are said to be *overfit*. Because these models capture the systematic component, predicted values from overfit models often have a small bias. However, because they also capture the random variation, they have a large variance. Large, complex models are often overfit.

Now consider a very simple model: $\hat{y} = c$, where c is a constant. The variance of the predicted value is 0, because it is constant. However. it is unlikely that the constant c will predict new observations well, on the average. Models that capture little of the systematic component are said to be *underfit*. Predictions from underfit models tend to have small variance, but large bias.

In summary, simplistic models tend to be underfit, and have large bias and small variance. As we make models larger and more complex, the bias will decrease and the variance will increase. A large complex model is likely to be overfit, and will have a small bias and a large variance. Therefore it is usually best to choose a model that is somewhere in between the simplest and most complex that we can think of.

When the true model is not assumed to be known, there are no formulas to compute the bias or the variance, and thus no algorithms to find an optimal predictor. Instead, we use methods that generate a collection of models of varying complexity, and estimate the MSEP of each with a computationally intensive algorithm such as cross-validation, to be discussed in Section 3.

2 Subset Selection Methods

We consider the problem of choosing a model in the following situation: We have a vector $\mathbf{Y} = (y_1, ..., y_n)^T$ of observations, and p potential predictor variables $\mathbf{x}_1, ..., \mathbf{x}_p$, with $\mathbf{x}_i = (x_{1i}, ..., x_n)^T$. We want to choose a subset of the predictor variables to define a linear model. By renumbering if necessary, assume we choose $\mathbf{x}_1, ..., \mathbf{x}_k$. Let $\mathbf{X}$ be the $n \times (k+1)$ matrix whose leftmost column is a column of 1s, for an intercept, and whose remaining columns are $\mathbf{X}_1, ..., \mathbf{X}_k$. Let $\boldsymbol{\beta} = (\beta_0, \beta_1, ..., \beta_p)^T$. The model is

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

We use the model to estimate $\boldsymbol{\beta}$ and to compute fitted values.

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

Recall that the statistic generally used to measure goodness-of-fit of a linear model is R^2:

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

The quantity $\sum_{i=1}^n (y_i - \hat{y}_i)^2$ is the sum of the squared residuals, which is sometimes called the residual sum of squares (RSS).

2.1 Best Subset Selection

The idea behind best subset selection is simple: First fit all models containing only one variable and pick the one that fits best, that is, the one with the largest value of R^2. Call this model M_1. Then fit all models containing two variables, and pick the one with the largest value of R^2. Call this model M_2. Then continue, fitting all models containing 3 variables, 4 variables, and so on, finishing with the model that contains all p variables. For each number of variables k, let M_k be the k-variable with the largest value of R^2. Best subsets selection thus produces a collection $M_1, ..., M_p$ of models, containing from 1 to p predictor variables.

Best subsets selection is computationally intensive when the number p of variables is large. There are a total of 2^p models to fit, so the method becomes infeasible when p is greater than about 40. A computationally less intensive but less comprehensive alternative to best subsets selection is stepwise selection, which we now describe.

2.2 Stepwise Selection

There are several variants of stepwise selection. We describe the simplest, which is called *forward* selection. Following are the steps:

- Choose the one-variable model with the largest value of R^2. Call it M_1. (This will be the same one-variable model chosen by best subsets).
- Consider all two-variable models formed by adding one variable to M_1. Choose the one with the largest value of R^2 and call it M_2.
- Continue in this fashion. Given the k-variable model M_k, consider all $k+1$-variable models formed by adding one variable to M_k. Choose the one with the largest value of R^2 and call it M_{k+1}.

Like best subsets selection, stepwise selection produces a collection $M_1, ..., M_p$ of models, containing from 1 to p predictor variables. It is computationally less intensive than best subsets, because at each step we consider only those models that contain all the variables in the previous model.

2.3 Formulas for Choosing a Single Model

Both best subsets and stepwise selection produce a collection of models, with numbers of variables ranging from 1 to p. We need a method to choose one of these models. Traditionally this has been done by calculating statistics that measure the goodness-of-fit.

The usual measure of goodness-of-fit, R^2, does not work well when comparing models with different numbers of variables. The reason is that R^2 measures how close the fitted values $\hat{\mathbf{Y}}$ are to the observed values $\mathbf{Y}$. However, we are not interested in predicting the values we have already observed — we want to predict future unobserved values. Because the observed values $\mathbf{Y}$ have been used to fit the model, the model will generally predict them more closely than it will predict unobserved values. Therefore measures of goodness of fit such as R^2, that describe how close $\hat{\mathbf{Y}}$ is to $\mathbf{Y}$, generally provide overoptimistic estimates of model performance. A particular problem with R^2 is that whenever a variable is added to the model, the value of R^2 always increases. Therefore, using R^2 as a criterion will always result in including every variable in the model. Various alternatives have been proposed that adjust for the number of variables in the model.

We describe four of these methods. Each is a statistic that contains a measure of goodness-of-fit (how close the fitted values are to the observations) along with a penalty term that adjusts for the number of variables in the model. In what follows n is the number of observations, p is the total number of variables available, k is the number of variables in the model under consideration, not counting the intercept, and RSS is the sum of squared residuals, $RSS = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$.

- Adjusted R^2:

$$\text{Adjusted } R^2 = R^2 - \left(\frac{k}{n-k-1}\right)(1-R^2)$$

  The adjusted R^2 is always smaller than R^2, since a positive quantity is subtracted from R^2. As the number of variables k increases, R^2 will increase, but the amount subtracted from it will increase as well. Larger values of adjusted R^2 indicate better-fitting models.

- The Akaike Information Criterion (AIC):

$$\text{AIC} = n \log\left(\frac{\text{RSS}}{n}\right) + 2(k+1)$$

  The principle is that as more variables are added to the model, the sum of squared residuals (RSS) will decrease, but the number of variables k in the model will increase. Smaller values of AIC indicate better-fitting models.

- The Bayesian Information Criterion (BIC):

$$\text{BIC} = n \log\left(\frac{\text{RSS}}{n}\right) + (k+1)\log n$$

  This statistic is also known as the Schwartz Bayesian Criterion (SBC). The principle is the same as that of AIC: as more variables are added to the model, the sum of squared residuals (RSS) will decrease, but the number of variables k in the model will increase. The penalty for adding another variable to the model is much higher for BIC: $\log n$ rather than 2. Smaller values of BIC indicate better-fitting models.

- Mallows' C_p:

$$C_p = \frac{(n-p-1)RSS_{reduced}}{RSS_{full}} - (n-2k-2)$$

  Here RSS_{full} is the sum of squared residuals from the model containing all p variables, and $\text{RSS}_{reduced}$ is the sum of squared residuals from the model under consideration,

which contains a subset of k variables. Smaller values of C_p indicate better-fitting models.

Of these four methods, BIC tends to choose the smallest models, because the penalty for adding a new variable is greatest. AIC and C_p give similar results, in fact, some sources present a version of C_p that is identical to AIC. Adjusted R^2 tends to choose models similar in size to those chosen by AIC and C_p.

2.4 An Example

We now illustrate these methods with a real data set. The file `College.csv` contains information on 777 colleges. Following is documentation.

Variable Name	Description
`Name`	Name of college
`Private`	Factor with levels No and Yes
`Apps`	Number of applications received
`Accept`	Number of applications accepted
`Enroll`	Number of new students enrolled
`Top10perc`	Percentage of new students from top 10% of H.S. class
`Top25perc`	Percentage of new students from top 25% of H.S. class
`F.Undergrad`	Number of full-time undergraduates
`P.Undergrad`	Number of part-time undergraduates
`Outstate`	Out-of-state tuition
`Room.Board`	Room and board costs
`Books`	Estimated book costs
`Personal`	Estimated personal spending
`PhD`	Percent of faculty with Ph.D.s
`Terminal`	Percent of faculty with terminal degree
`S.F.Ratio`	Student/faculty ratio
`perc.alumni`	Percentage of alumni who donate
`Expend`	Instructional expenditure per student
`Grad.Rate`	Graduation rate

We will construct a model to predict `Apps`, the number of applications received. We will use the best subsets method, as implemented in R. The command in R is `regsubsets`, which is in the `leaps` library. We begin by loading this library and reading in the data set.

```
> library(leaps)
> coll = read.csv("College.csv", header=TRUE)
```

```
> coll = coll[,-1]                                  #Remove names of colleges
> coll$Private = as.numeric(coll$Private) - 1   #Change "Yes" "No" to numeric

#Find the best model of each size

> best = regsubsets(Apps~., data=coll)
> best.sum = summary(best)

#By default, regsubsets finds the best models of sizes 1 through 8. We now
#specify which variables are in these models.

> best.sum$which
  (Intercept) Private Accept Enroll Top10perc Top25perc F.Undergrad P.Undergrad
1        TRUE   FALSE   TRUE  FALSE     FALSE     FALSE       FALSE       FALSE
2        TRUE   FALSE   TRUE  FALSE      TRUE     FALSE       FALSE       FALSE
3        TRUE   FALSE   TRUE  FALSE      TRUE     FALSE       FALSE       FALSE
4        TRUE   FALSE   TRUE  FALSE      TRUE     FALSE       FALSE       FALSE
5        TRUE   FALSE   TRUE   TRUE      TRUE     FALSE       FALSE       FALSE
6        TRUE   FALSE   TRUE   TRUE      TRUE     FALSE       FALSE       FALSE
7        TRUE   FALSE   TRUE   TRUE      TRUE      TRUE       FALSE       FALSE
8        TRUE    TRUE   TRUE   TRUE      TRUE     FALSE       FALSE       FALSE
  Outstate Room.Board Books Personal   PhD Terminal S.F.Ratio perc.alumni
1    FALSE      FALSE FALSE    FALSE FALSE    FALSE     FALSE       FALSE
2    FALSE      FALSE FALSE    FALSE FALSE    FALSE     FALSE       FALSE
3    FALSE      FALSE FALSE    FALSE FALSE    FALSE     FALSE       FALSE
4     TRUE      FALSE FALSE    FALSE FALSE    FALSE     FALSE       FALSE
5     TRUE      FALSE FALSE    FALSE FALSE    FALSE     FALSE       FALSE
6     TRUE       TRUE FALSE    FALSE FALSE    FALSE     FALSE       FALSE
7     TRUE       TRUE FALSE    FALSE FALSE    FALSE     FALSE       FALSE
8     TRUE       TRUE FALSE    FALSE  TRUE    FALSE     FALSE       FALSE
  Expend Grad.Rate
1  FALSE     FALSE
2  FALSE     FALSE
3   TRUE     FALSE
4   TRUE     FALSE
5   TRUE     FALSE
6   TRUE     FALSE
7   TRUE     FALSE
8   TRUE     FALSE
```

We can see from the output above that the best 8-variable model contains the variables Private, Accept, Enroll, Top10perc, Outstate, Room.Board, PhD, and Expend.

There are a total of 17 available predictor variables. We would like to find the best model for each number of variables up to 17. This can be done by including the argument

`nvmax = 17` in the `regsubsets` command. We then compute Adjusted R^2, BIC, and C_p for each of the 17 models. We don't compute AIC because it is very similar to C_p.

```
> best = regsubsets(Apps~., nvmax=17, data=coll)
> best.sum = summary(best)

> cbind(best.sum$rsq, best.sum$adjr2, best.sum$bic, best.sum$cp)

           [,1]      [,2]      [,3]      [,4]
 [1,] 0.8900990 0.8899572 -1702.441 404.98808
 [2,] 0.9157839 0.9155663 -1902.619 131.68051
 [3,] 0.9183356 0.9180186 -1919.870 106.33057
 [4,] 0.9212640 0.9208560 -1941.588  76.94215
 [5,] 0.9237599 0.9232655 -1959.962  52.18943
 [6,] 0.9247464 0.9241600 -1963.427  43.61538
 [7,] 0.9257649 0.9250892 -1967.359  34.69833
 [8,] 0.9268725 0.9261108 -1972.384  24.82608
 [9,] 0.9276780 0.9268294 -1974.335  18.19266
[10,] 0.9283103 0.9273744 -1974.502  13.41556
[11,] 0.9288011 0.9277773 -1973.184  10.15485
[12,] 0.9289945 0.9278792 -1968.643  10.08161
[13,] 0.9291223 0.9279147 -1963.387  10.71131
[14,] 0.9291632 0.9278617 -1957.180  12.27345
[15,] 0.9291878 0.9277921 -1950.795  14.00926
[16,] 0.9291885 0.9276978 -1944.147  16.00190
[17,] 0.9291887 0.9276027 -1937.493  18.00000
```

We first point out that the values of BIC don't match the formula given above. What R does is compute the value of BIC for each model, then subtract from it the value of BIC for the model containing only an intercept. This produces rather large negative numbers. However, the model selected is the one whose value of BIC is the least.

The output informs us that Adjusted R^2 chooses the 13-variable model, BIC chooses the 10-variable model, and C_p chooses the 12-variable model. We'll find out which variables are in each of these models.

```
#Model chosen by Adjusted R^2
> coef(best.sum$obj, 13)
  (Intercept)       Private        Accept        Enroll     Top10perc
-440.74148270 -484.77261885    1.58542302   -0.87824288   50.41461998
    Top25perc   F.Undergrad   P.Undergrad      Outstate    Room.Board
 -14.63667155    0.05762769    0.04642270   -0.08823311    0.14696204
```

```
          PhD     S.F.Ratio        Expend     Grad.Rate
 -10.91804823   15.15475056    0.07786425    8.58578735

#Model chosen by BIC
> coef(best.sum$obj, 10)
  (Intercept)       Private        Accept        Enroll     Top10perc
-100.51668243 -575.07060789    1.58421887   -0.56220848   49.13908916
    Top25perc      Outstate    Room.Board           PhD        Expend
 -13.86531103   -0.09466457    0.16373674  -10.01608705    0.07273776
    Grad.Rate
   7.33268904

#Model chosen by C_p
> coef(best.sum$obj, 12)
  (Intercept)       Private        Accept        Enroll     Top10perc
-157.28685883 -511.78760196    1.58691470   -0.88265385   50.41131660
    Top25perc   F.Undergrad   P.Undergrad      Outstate    Room.Board
 -14.74735373    0.05945481    0.04593068   -0.09017643    0.14776586
          PhD        Expend     Grad.Rate
 -10.70502848    0.07246655    8.63961002
```

Here is the same example using stepwise regression:

```
> step = regsubsets(Apps~., nvmax=17, method="forward", data=coll)
> step.sum = summary(step)

> cbind(step.sum$rsq, step.sum$adjr2, step.sum$bic, step.sum$cp)
           [,1]      [,2]      [,3]      [,4]
 [1,] 0.8900990 0.8899572 -1702.441 404.98808
 [2,] 0.9157839 0.9155663 -1902.619 131.68051
 [3,] 0.9183356 0.9180186 -1919.870 106.33057
 [4,] 0.9212640 0.9208560 -1941.588  76.94215
 [5,] 0.9237599 0.9232655 -1959.962  52.18943
 [6,] 0.9247464 0.9241600 -1963.427  43.61538
 [7,] 0.9257649 0.9250892 -1967.359  34.69833
 [8,] 0.9267531 0.9259902 -1971.117  26.10571
 [9,] 0.9276780 0.9268294 -1974.335  18.19266
[10,] 0.9283103 0.9273744 -1974.502  13.41556
[11,] 0.9288011 0.9277773 -1973.184  10.15485
[12,] 0.9289945 0.9278792 -1968.643  10.08161
[13,] 0.9291223 0.9279147 -1963.387  10.71131
[14,] 0.9291632 0.9278617 -1957.180  12.27345
[15,] 0.9291878 0.9277921 -1950.795  14.00926
[16,] 0.9291885 0.9276978 -1944.147  16.00190
[17,] 0.9291887 0.9276027 -1937.493  18.00000
```

In this case, the models chosen by stepwise regression are the same as the ones chosen by best subsets, except for the 8-variable model. The 8-variable model chosen by stepwise selection contains the variables Private, Accept, Enroll, Top10perc, Top25perc, Outstate, Room.Board, and Expend. This differs from the one chosen by best subsets by including Top25perc in place of PhD. The stepwise models chosen by Adjusted R^2, BIC, and C_p are the same as the ones chosen by best subsets selection.

3 Cross-Validation

3.1 Training Sets and Test Sets

Statistics such as adjusted R^2, AIC, BIC, and C_p estimate MSEP with formulas that estimate goodness-of-fit along with a penalty for the number of variables in the model. The ideal way to estimate MSEP would be to collect some new data after the model has been constructed, and calculate the prediction errors obtained when the model is used to predict these values.

In practice, we are rarely able to collect new data, but we can employ the concept of a *training set* and a *test set*, which mimics the estimation of MSEP from a new data set. The data are randomly divided into two pieces, one designated as the training set and the other as the test set. A model selection procedure is performed on the training set. Then the chosen model is used to predict the observations in the test set. The MSEP of the test set is an estimate of the MSEP on new observations. Since the test set is not used in the model selection procedure, it plays the role of new data.

3.2 Test Sets Aren't As Good As Actual New Data

Withholding a test set and using it to represent new data is often the best available method to evaluate the performance of a model on new data. However it is not as good as evaluating the model on new data. The training and test sets are randomly chosen from the same sample, thus there are no systematic differences between them. In reality, future data will often be generated under somewhat different circumstances than the data used to construct a model. Thus the prediction errors on new data may tend to be larger than those on the test set.

This is an important problem, and there are no general methods to address it. Data analysts must use judgment to anticipate and understand potential differences between present and future data sets, and to adjust predictions accordingly. Model selection is an art, not a science.

Another problem, which can be solved, is that the results of the training set/test set approach depend on the random division into training and test sets. The estimate of the prediction error can therefore be highly variable. This problem is addressed by use of a method known as *cross-validation*. The basic idea is that one splits the data into training and test sets in many different ways and averages the results together. The first method we discuss is

leave-one-out-cross-validation (LOOCV).

3.3 Leave-One-Out-Cross-Validation

In leave-one-out-cross-validation (LOOCV), a single observation is held out as a test set, and the rest of the data is used to construct a model. That model is used to predict the held-out value, and the prediction error is computed. This is repeated for every observation in the data set, and the squared prediction errors are averaged. We present a step-by-step description. Let n be the number of observations in the entire data set, and denote the observations by $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)$. where $\mathbf{x}_i$ is a vector of predictor variables used to predict y_i.

For $i = 1, ..., n$:

1. Hold out observation $(\mathbf{x}_i, y_i)$.
2. Perform a model selection procedure using all observations except $(\mathbf{x}_i, y_i)$.
3. Let $\hat{y}_i$ be the model prediction of y_i, using the value $\mathbf{x}_i$.
4. Compute the squared prediction error $(\hat{y}_i - y_i)^2$.

Then compute MSEP $= \sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}$.

We present an application to the College data set. The goal is to construct a model to predict the number of applications, `Apps`, from the 17 other variables in the data set. The model selection procedure is best subsets.

We begin by setting the random seed (`set.seed(1)`) so the results will be reproducible.

```
> set.seed(1)
> library(leaps)
> coll = read.csv("College.csv", header=TRUE)
> coll = coll[,-1]
> coll$Private = as.numeric(coll$Private) - 1

> n = nrow(coll)

#The j,i element of the following matrix will contain the squared prediction
#error for the jth observation, using the best i-variable model chosen from
#the remaining observations. Averaging down the columns will produce MSEP for
```

```
#each size model.
> cv.errors = matrix(NA, n, 17, dimnames=list(NULL, c(1:17)))

> for (j in 1:n){

#Hold out the jth observation and compute the best models of sizes 1-17.
#Note that the rows are designated by -j.
+   best.fit = regsubsets(Apps ~., data=coll[-j,], nvmax = 17)

#Find the best model of each size, holding out the jth observation
+   for (i in 1:17){

#The model.matrix command adds an intercept to the matrix specified in the data
option. This is a single row: the jth row, which was withheld.
+   testmat = model.matrix(Apps ~ ., data = coll[j,])

#Extract the coefficients from the best model with i variables.
+   coefi = coef(best.fit, id=i)

#Get a list of the names of these variables (including "Intercept").
+   xvars = names(coefi)

#Compute the predicted value for the jth observation
+   pred = testmat[,xvars]%*%coefi

#Compute the squared prediction error for the jth observation from the best
#model of i variables.
+   cv.errors[j, i] = (coll$Apps[j] - pred)^2
+   }
+  }

#Compute the mean of each column of the matrix cv.errors. ("2" represents
#columns.)
>  msep = apply(cv.errors, 2, mean)
>  msep
      1       2       3       4       5       6       7       8       9      10
1695819 1330246 1321653 1325924 1314869 1382564 1319539 1273813 1260216 1284884
11      12      13      14      15      16      17
1265007 1273287 1275967 1274523 1273194 1276669 1276987
```

The MSEP for the 9-variable models is the minimum. We will therefore find the best 9-variable model, using all the data.

```
> best9 = regsubsets(Apps ~., data=coll, nvmax = 9)
> coef(best9, id=9)
```

```
  (Intercept)       Private        Accept        Enroll     Top10perc
123.04301307 -536.82577612    1.59312461   -0.57792891   50.46142370
   Top25perc      Outstate    Room.Board           PhD        Expend
-12.91493827   -0.08418409    0.17306218   -9.80197575    0.06940960
```

The model chosen by best subsets and LOOCV is the 9-variable model with the variables Private, Accept, Enroll, Top10perc, Top25perc Outstate, Room.Board, PhD, and Expend.

LOOCV is computationally intensive for large data sets, because the model selection procedure has to be performed once for every observation in the data set. An alternative is k-fold cross-validation, which we now describe.

3.4 k-fold Cross-validation

In k-fold cross-validation, the data are randomly divided into k approximately equal parts, called "folds." The first fold is held out, and a model selection procedure is performed on the remaining data. The chosen model is used to predict the values in the held-out fold, and the mean squared prediction error, MSEP_1, is calculated on the observations in the held-out fold. The procedure is repeated k times, each time holding out a different fold. This produces k estimates of MSEP: MSEP_1, ..., MSEP_k. The k-fold cross-validation estimate of MSEP is the average of MSEP_1, ..., MSEP_k.

Here is the step-by-step procedure:

1. Randomly split the data into k folds. Let $y_{j1}, ..., y_{jm}$ be the observed values in the jth fold.
2. For $j = 1, ..., k$:
   - Hold out fold j and perform a model selection procedure on all the data except that in fold j.
   - Use the chosen model to predict the values in fold j.
   - Compute the mean squared prediction error for fold j: $\text{MSEP}_j = \dfrac{\sum_{i=1}^{m}(\hat{y}_{ji} - y_{ji})^2}{m}$.
3. Estimate MSEP with the average of MSEP_1, ..., MSEP_k: $\text{MSEP} = \dfrac{\sum_{j=1}^{k}\text{MSEP}_j}{k}$.

The most commonly used values for k are $k = 5$ and $k = 10$.

Note that LOOCV is a special case of k-fold cross-validation where k is equal to the number of observations in the data set.

Advantages of k-fold cross-validation over LOOCV

The obvious advantage of k-fold cross-validation over LOOCV is that it requires less computation. Interestingly enough, k-fold cross-validation often gives more accurate estimates of MSEP. This results from the bias-variance trade-off. LOOCV has a very small bias, because the models are chosen on the basis of sets containing all but one of the observations. However, somewhat surprisingly, the variance of LOOCV is higher than that of cross-validation with fewer folds. This is caused by the fact that the training sets for LOOCV are almost identical to each other. For this reason, the estimates of MSEP that are averaged together have a high positive correlation, which results in a large variance. As a result, k-fold cross-validation with $k = 5$ or $k = 10$ often provides more accurate estimates of MSEP than does LOOCV.

An example

We now present an example in which use the College data set to construct a model to predict the number of applications from the other 17 variables using best subsets selection and 10-fold cross-validation.

We begin by setting the random seed (`set.seed(1)`) so the results will be reproducible.

```
> set.seed(1)
> k = 10
> folds = sample(1:k, nrow(coll), replace=TRUE)
> cv.errors = matrix(NA, k, 17, dimnames=list(NULL, c(1:17)))
>
>  for (j in 1:k){
+   best.fit = regsubsets(Apps ~., data=coll[folds!=j,], nvmax = 17)
+   testmat = model.matrix(Apps ~ ., data = coll[folds==j,])
+   for (i in 1:17){
+   coefi = coef(best.fit, id=i)
+   xvars = names(coefi)
+   pred = testmat[,xvars]%*%coefi
+   cv.errors[j, i] = mean((coll$Apps[folds==j] - pred)^2)
+   }
+  }
>
> msep = apply(cv.errors, 2, mean)
```

```
> msep
      1       2       3       4       5       6       7       8       9      10
1676826 1315700 1310869 1307628 1294688 1296410 1296005 1270474 1242055 1259176
     11      12      13      14      15      16      17
1271657 1276103 1270017 1271545 1272634 1275335 1275865
```

The MSEP for the 9-variable models is the minimum. We will therefore find the best 9-variable model, using all the data.

```
> best9 = regsubsets(Apps~., nvmax=9, data=coll)
> coef(best9, id=9)
  (Intercept)       Private        Accept        Enroll     Top10perc
 123.04301307 -536.82577612    1.59312461   -0.57792891   50.46142370
    Top25perc      Outstate    Room.Board           PhD        Expend
 -12.91493827   -0.08418409    0.17306218   -9.80197575    0.06940960
```

The model chosen by best subsets and 10-fold cross validation is the same as the one chosen by LOOCV.

4 Shrinkage Methods

Shrinkage methods for fitting linear models have the effect of reducing the magnitude of the estimated coefficients, in other words, shrinking them towards zero. This increases bias, but often can lower the variance enough to provide an overall reduction in MSEP. We focus on two methods: ridge regression and the lasso.

4.1 Ridge Regression

Linear regression estimates are the values $\hat{\beta}_0, \hat{\beta}_1, ..., \hat{\beta}_p$ that minimize the sum of squares

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2$$

The ridge regression estimates minimize

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

where $\lambda \geq 0$ is a "tuning parameter," whose value is chosen by the data analyst. The term $\lambda\sum_{j=1}^{p}\beta_j^2$ is sometimes called a "penalty term." Its effect is to reduce the magnitude of the estimates $\hat{\beta}_j$.

Note that the intercept β_0 is not involved in the penalty. We don't want to shrink the intercept, because it simply indicates the mean of the y_i when the predictor variables are set to 0. The impact of the penalty depends on the value of λ. When $\lambda = 0$, there is no penalty, and ridge regression is the same as linear regression. When λ is small, the ridge regression estimates will be close to the linear regression estimates. When λ is large, the ridge regression estimates will be forced to be close to 0. In practice, the value of λ is usually chosen by cross-validation; we will discuss this later.

Ridge regression can be defined as a constrained minimization problem, as the following theorem shows:

Theorem 4.1. Let $\lambda \geq 0$, and let $\hat{\boldsymbol{\beta}}$ be the ridge regression estimates, minimizing $\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\sum_{j=1}^{p}\beta_j^2$. Then there exists a non-negative constant γ such that $\hat{\boldsymbol{\beta}}$ minimizes $\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2$ subject to the constraint $\sum_{j=1}^{p}\beta_j^2 \leq \gamma$.

Proof:
Let $\hat{\boldsymbol{\beta}}$ be the ridge regression estimates. Let $\gamma = \sum_{j=1}^{p} \hat{\beta}_j^2$. We have thus chosen γ so that $\hat{\boldsymbol{\beta}}$ satisfies the constraint. Let $\boldsymbol{\beta}^*$ be any vector satisfying the constraint $\sum_{j=1}^{p} \beta_j^{*2} \leq \gamma$. Then $\sum_{j=1}^{p} \beta_j^{*2} \leq \sum_{j=1}^{p} \hat{\beta}_j^2$. Since $\hat{\boldsymbol{\beta}}$ minimizes $\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} \beta_j^2$,

$$\sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \sum_{j=1}^{p} \hat{\beta}_j x_{ij})^2 + \lambda \sum_{j=1}^{p} \hat{\beta}_j^2 \leq \sum_{i=1}^{n}(y_i - \beta_0^* - \sum_{j=1}^{p} \beta_j^* x_{ij})^2 + \lambda \sum_{j=1}^{p} \beta_j^{*2} \tag{4.1}$$

Since $\sum_{j=1}^{p} \hat{\beta}_j^2 = \gamma$ and $\sum_{j=1}^{p} \beta_j^{*2} \leq \gamma$, it follows that $\sum_{j=1}^{p} \beta_j^{*2} \leq \sum_{j=1}^{p} \hat{\beta}_j^2$. Combining this with (4.1) shows that

$$\sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \sum_{j=1}^{p} \hat{\beta}_j x_{ij})^2 \leq \sum_{i=1}^{n}(y_i - \beta_0^* - \sum_{j=1}^{p} \beta_j^* x_{ij})^2$$

Since $\hat{\boldsymbol{\beta}}$ satisfies the constraint $\sum_{j=1}^{p} \hat{\beta}_j^2 \leq \gamma$, it follows that $\hat{\boldsymbol{\beta}}$ minimizes $\sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \sum_{j=1}^{p} \hat{\beta}_j x_{ij})^2$ subject to this constraint.

The ridge regression estimates can also be expressed in closed form as a linear combination of the observations $y_1, ..., y_n$, as the following theorem shows.

Theorem 4.2. Let $\mathbf{x}_1, ..., \mathbf{x}_p$ be predictor variables, and for $j = 1, ..., p$, let $\bar{x}_{\cdot j} = \dfrac{1}{n}\sum_{i=1}^{n} x_{ij}$. Let $\mathbf{X}$ be the $n \times p$ matrix whose ij element is $x_{ij} - \bar{x}_{\cdot j}$. Note that each column of $\mathbf{X}$ sums to 0. Let $\mathbf{I}$ be the $p \times p$ identity matrix. Let $\boldsymbol{\beta} = (\beta_1, ..., \beta_p)^T$. Note that the intercept is not included in $\boldsymbol{\beta}$. Then for a given value λ, the ridge regression estimate $\hat{\boldsymbol{\beta}}$ is given by

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}$$

Proof:
The ridge regression estimate $(\hat{\beta}_0, \hat{\beta}_1, ..., \hat{\beta}_p)^T$ minimizes

$$S = \sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij}\right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

The partial derivatives with respect to β_k are therefore equal to 0 for $k = 0, ..., p$. We begin with the partial derivative with respect to β_0.

$$\frac{\partial S}{\partial \beta_0} = \sum_{i=1}^{n} -2\left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij}\right) = 0$$

It follows that

$$\hat{\beta}_0 = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{p}\hat{\beta}_j x_{ij}\right) = \bar{y} - \sum_{j=1}^{p}\hat{\beta}_j \bar{x}_{.j}$$

Substituting into the expression for S yields

$$S = \sum_{i=1}^{n}\left(y_i - \bar{y} - \sum_{j=1}^{p}(x_{ij} - \bar{x}_{.j})\beta_j\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

Now let $x_{ij}^* = x_{ij} - \bar{x}_{.j}$. Note that $\sum_{i=1}^{n} x_{ij}^* = 0$. We now have

$$S = \sum_{i=1}^{n}\left(y_i - \bar{y} - \sum_{j=1}^{p}\beta_j x_{ij}^*\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

Now let $\mu_i = \sum_{j=1}^{p}\beta_j x_{ij}^*$. Note that $\dfrac{\partial \mu_i}{\partial \beta_k} = x_{ik}^*$. We can write the sum of squares S as

$$S = \sum_{i=1}^{n}(y_i - \bar{y} - \mu_i)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

The derivative with respect to β_k for $k = 1, ..., p$ is

$$\frac{\partial S}{\partial \beta_k} = -2\sum_{i=1}^{n}(y_i - \bar{y} - \mu_i)\frac{\partial \mu_i}{\partial \beta_k} + 2\lambda\beta_k$$

Since $\dfrac{\partial \mu_i}{\partial \beta_k} = x_{ik}^*$, we compute $\hat{\boldsymbol{\beta}}$ by solving the system of equations

$$\sum_{i=1}^{n}(y_i - \bar{y} - \mu_i)\, x_{ik}^* - \lambda\beta_k = 0 \qquad \text{for } k = 1, ..., p$$

Multiplying out, we have

$$\sum_{i=1}^{n}(y_i - \bar{y} - \mu_i)\, x_{ik}^* - \lambda\beta_k = \sum_{i=1}^{n} y_i x_{ik}^* - \bar{y}\sum_{i=1}^{n} x_{ik}^* - \sum_{i=1}^{n}\mu_i x_{ik}^* - \lambda\beta_k$$

Since $\sum_{i=1}^{n} x_{ik}^* = 0$, the system of equations becomes

$$\sum_{i=1}^{n} y_i x_{ik}^* - \sum_{i=1}^{n}\mu_i x_{ik}^* - \lambda\beta_k = \sum_{i=1}^{n} x_{ik}^* y_i - \sum_{i=1}^{n} x_{ik}^*\mu_i - \lambda\beta_k = 0 \qquad \text{for } k = 1, ..., p$$

Now let $\mathbf{x}_k = (x_{1k}^*, ..., x_{nk}^*)^T$. Notice that $\mathbf{x}_k$ is the kth column of $\mathbf{X}$. Writing the system of equations as dot products yields

$$\mathbf{x}_k^T\mathbf{Y} - \mathbf{x}_k^T\boldsymbol{\mu} - \lambda\beta_k = 0 \qquad \text{for } k = 1, ..., p$$

In matrix form, this is

$$\mathbf{X}^T\mathbf{Y} - \mathbf{X}^T\boldsymbol{\mu} - \lambda\boldsymbol{\beta} = \mathbf{0}$$

Now substitute $\boldsymbol{\mu} = \mathbf{X}\boldsymbol{\beta}$, and let $\mathbf{I}$ be the $p \times p$ identity matrix. We now have

$$\mathbf{X}^T\mathbf{Y} - \mathbf{X}^T\mathbf{X}\boldsymbol{\beta} - \lambda\mathbf{I}\boldsymbol{\beta} = \mathbf{X}^T\mathbf{Y} - (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})\boldsymbol{\beta} = \mathbf{0}$$

Solving for $\boldsymbol{\beta}$, we have

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}$$

We can see that the ridge regression estimates differ from ordinary least squares by the addition of the term $\lambda\mathbf{I}$. This adds the constant λ to each diagonal element of $\mathbf{X}^T\mathbf{X}$. This is the "ridge" that gives ridge regression its name.

Standardizing the Predictors

Linear regression estimates have a useful property known as *scale invariance*. If a predictor variable is multiplied by a constant c, its coefficient will be divided by c, so that the predicted values will remain the same. So, for example, if one of our variables is length measured in feet and we change the units to inches (multiplying by 12), the coefficient of length will be divided by 12. As a result, predictions made by linear regression do not depend on the units used for the variables.

Ridge regression estimates, unfortunately, are not scale invariant. If we change the units of one of the variables, all the coefficients may change their values, and the predicted values will change as well. For this reason, variables should be *standardized* before ridge regression is performed. Standardization is accomplished simply by dividing each predictor variable by its standard deviation. For each j let $\bar{x}_{\cdot j}$ be the average of $x_{1j}...., x_{nj}$. The standardized version of x_{ij} is

$$\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{\sum_{i=1}^{n}(x_{ij} - \bar{x}_{\cdot j})^2/n}}$$

The command in R usually used for ridge regression automatically standardizes the variables, so you don't have to do it yourself.

An Example

We use ridge regression to predict expenditures (`Expend`) using the College data set. Following is R code:

```
> library{glmnet}
> x = model.matrix(Expend~., data=coll)[,-1]
> y = coll$Expend
> grid = 10^seq(-2, 8, length=100)
> ridge = glmnet(x, y, alpha = 0, lambda=grid)
```

We explain each line of code. The first line loads the necessary library. The command `x = model.matrix(Expend~., data=coll)[,-1]` creates a matrix suitable for fitting a linear model to predict the response `Expend`. The first column of `x` is an intercept, and the rest of matrix consists of all the columns of **coll** except `Expend`. We don't want `x` to contain an intercept, because the `glmnet` command will include one automatically. The `[,-1]` has the effect of deleting the first column, which is the intercept.

The command `y = coll$Expend` creates the response variable.

The command `grid = 10^seq(-2, 8, length=100)` creates a vector of 100 values between 10^{-2} and 10^8.

The command `ridge = glmnet(x, y, alpha = 0, lambda=grid)` performs ridge regression for every value of λ in the vector `grid`. There are four arguments: `x` is a matrix containing the predictor variables, and `y` is the response variable. The command `alpha = 0` specifies that ridge regression is to be performed. Finally, `lambda = grid` specifies the values of λ that are to be used.

Interestingly enough, once `glmnet` is run for a large range of values of λ, we can use a different command, `predict`, to get the coefficients for a single value of λ. Note that this value does not have to be one of the ones specified in the `glmnet` command. We compute the coefficients for $\lambda = 1$, 100, and 1000.

```
> coef1 = predict(ridge, s = 1, type="coefficients")
> coef2 = predict(ridge, s = 100, type="coefficients")
> coef3 = predict(ridge, s = 1000, type="coefficients")
> coef1 = as.matrix(coef1)
> coef2 = as.matrix(coef2)
> coef3 = as.matrix(coef3)
> cbind(coef1, coef2, coef3)

(Intercept) 7147.10918394   6790.0667353   4333.32787890
Private     -659.63860189   -628.2444776   -133.87144071
Apps           0.63139448     0.46525034      0.20596210
```

```
Accept        -0.89819791    -0.55450166     -0.09744065
Enroll         0.73477371     0.31528863      0.03192136
Top10perc    129.42466279   125.07981007     80.45903762
Top25perc    -61.25825231   -51.64390614     -7.83345339
F.Undergrad   -0.06092318    -0.03324668     -0.01649086
P.Undergrad    0.03685317     0.03930096      0.02491865
Outstate       0.46140811     0.42959902      0.31740025
Room.Board     0.29219584     0.33064867      0.43879270
Books          0.70394781     0.72657379      0.79658278
Personal       0.18987375     0.20436704      0.19674833
PhD           12.82591430    13.20106046     17.01002920
Terminal      18.71213091    17.25002731     17.11454559
S.F.Ratio   -377.20668161  -378.91042646   -346.79221180
perc.alumni    7.73782091    10.48973541     17.49883218
Grad.Rate    -24.95631859   -22.99452601    -12.24976785

#Compute squared lengths of coefficient vectors, excluding intercepts
> cbind(sum(coef1[-1]^2, coef2[-1]^2, coef3[-1]^2))

         [,1]     [,2]     [,3]
[1,] 599111.2 557688.4 145671.0
```

The larger the value of λ, the smaller the coefficients.

Choosing the Tuning Parameter with Cross-Validation

The tuning parameter λ is usually chosen by cross-validation. Here are the steps for k-fold cross-validation. For LOOCV, just set $k = n$, the number of observations in the data set.

1. Specify a grid of λ values, spanning a wide range.
2. Randomly divide the data into k folds.
3. For each λ in the grid
   - For $j = 1, ..., k$
     - Fit a ridge regression model, using the value of λ, to all the data except that in fold j
     - Use the fitted model to predict the response values in fold j.
     - Compute the mean squared prediction error for fold j: $\text{MSEP}(\lambda)_j = \dfrac{\sum_{i=1}^{m}(\hat{y}_{ji} - y_{ji})^2}{m}$.
   - Compute $\text{MSEP}(\lambda) = \dfrac{\sum_{j=1}^{k}\text{MSEP}_j}{k}$.

We apply 10-fold cross-validation to choose a model to predict `Expend` from the other variables in the College data set.

We begin by setting the random seed (`set.seed(1)`) so the results will be reproducible.

```
> set.seed(1)
> k = 10
> grid = 10^seq(-2, 10, length=100)
> folds = sample(1:k, nrow(coll), replace=TRUE)
> cv.errors = matrix(NA, k, 100, dimnames=list(NULL, c(1:100)))
>
> for (j in 1:k){
+  ridge = glmnet(x[folds!=j,], y[folds!=j], alpha = 0, lambda=grid)
+   testmat = model.matrix(Expend~., data=coll[folds==j,])
+  for (i in 1:100){
+   coefi = coef(ridge)[,i]
+   pred = testmat%*%coefi
+   cv.errors[j, i] = mean((coll$Expend[folds==j] - pred)^2)
+  }
+ }
>
> msep = apply(cv.errors, 2, mean)
> msepse = apply(cv.errors, 2, sd)/sqrt(k)
```

The array `msep` contains the estimated MSEP for each value of λ, and the array `msepse` contains the standard deviation of the MSEP estimates for each of the 10 folds. We now find the minimum MSEP.

```
> min(msep)
[1] 9575724
> which.min(msep)
70
```

The minimum MSEP is 9575724, and it corresponds to the 70th value of λ tested. Now, the `glmnet` command applies the values of lambda in the vector `grid` in reverse order, in other words, the first value of λ tested is `grid[100]` and the 100th value tested is `grid[1]`. It follows that the 70th value tested has index $101 - 70 = 31$. So the value of λ corresponding to the minimum MSEP is `grid[31]`.

```
> lambda.min = grid[31]
> lambda.min
[1] 43.28761
```

We'll now fit the model with $\lambda = 43.28761$ to the whole data set. We'll first do it the way that seems obvious, then we'll do it a more cumbersome way which is slightly more accurate. The obvious way is to fit the single model with the minimum value of lambda.

```
> ridge.best = glmnet(x, y, alpha=0, lambda = lambda.min)
> coef(ridge.best)
18 x 1 sparse Matrix of class "dgCMatrix"
                      s0
(Intercept) 7008.51544990
Private     -654.17991270
Apps           0.54091071
Accept        -0.70875199
Enroll         0.48613275
Top10perc    128.71238092
Top25perc    -57.36256467
F.Undergrad   -0.04321169
P.Undergrad    0.03855026
Outstate       0.44564553
Room.Board     0.31103925
Books          0.71454193
Personal       0.19778721
PhD           12.85812477
Terminal      17.97168890
S.F.Ratio   -378.46522036
perc.alumni    9.16821582
Grad.Rate    -24.02617638
```

The following way is a bit more accurate. Fit the model to the grid of values, then use the predict command to compute coefficients for the value of interest.

```
> ridge.full = glmnet(x, y, alpha=0, lambda = grid)
> predict(ridge.full, s=lambda.min, type="coefficients")
18 x 1 sparse Matrix of class "dgCMatrix"
                       1
(Intercept) 7009.18166696
Private     -654.18898502
Apps           0.54072210
Accept        -0.70790975
Enroll         0.48313275
Top10perc    128.73143281
Top25perc    -57.37255093
F.Undergrad   -0.04291238
P.Undergrad    0.03854907
Outstate       0.44561356
Room.Board     0.31095756
Books          0.71459555
Personal       0.19784371
```

```
PhD           12.85914563
Terminal      17.96929347
S.F.Ratio   -378.47122693
perc.alumni    9.17832723
Grad.Rate    -24.02575226
```

4.2 The Lasso

The lasso, which is a acronym for Least Absolute Shrinkage and Selection Operator, is a shrinkage method that is superficially similar to ridge regression. The lasso estimates minimize the quantity

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p}|\beta_j|$$

The difference between the lasso and ridge regression is in the penalty term, which is $\lambda \sum_{j=1}^{p}\beta_j^2$ for ridge regression and $\lambda \sum_{j=1}^{p}|\beta_j|$ for the lasso. As in ridge regression, the intercept β_0 is not included in the penalty term.

As in ridge regression, the lasso can be defined as a constrained optimization problem, as follows.

Theorem 4.3. Let $\lambda \geq 0$, and let $\hat{\boldsymbol{\beta}}$ be the lasso estimates, minimizing $\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p}|\beta_j|$. Then there exists a non-negative constant γ such that $\hat{\boldsymbol{\beta}}$ minimizes $\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2$ subject to the constraint $\sum_{j=1}^{p}|\beta_j| \leq \gamma$.

Proof: The proof is similar to that of Theorem 4.1.

An Example

We use the lasso to predict expenditures (`Expend`) using the College data set. The R code is the same as for ridge regression except that `alpha = 0` is replaced with `alpha = 1`.

```
> library(glmnet)
> x = model.matrix(Expend~., data=coll)[,-1]
> y = coll$Expend
> grid = 10^seq(-2, 8, length=100)
> lasso = glmnet(x, y, alpha = 1, lambda=grid)
```

Analogous to ridge regression, this code performs the lasso for every value of λ in the array `grid`.

As with ridge regression, once `glmnet` has fit the lasso for a large range of values of λ, the `predict` command can be used to get the coefficients for any value of λ. Note that this value does not have to be one of the ones specified in the `glmnet` command. We compute the coefficients for $\lambda = 1$, 100, and 1000.

```
> coef1 = predict(lasso, s = 1, type="coefficients")
> coef2 = predict(lasso, s = 100, type="coefficients")
> coef3 = predict(lasso, s = 1000, type="coefficients")
> coef1 = as.matrix(coef1)
> coef2 = as.matrix(coef2)
> coef3 = as.matrix(coef3)
> cbind(coef1, coef2, coef3)
                         1             1             1
(Intercept) 7156.56443990  6.538630e+03  6.944336e+03
Private     -652.02296014 -1.426845e+02  0.000000e+00
Apps           0.62785863  1.655599e-01  0.000000e+00
Accept        -0.88698150 -3.452337e-03  0.000000e+00
Enroll         0.67620810  0.000000e+00  0.000000e+00
Top10perc    129.63451297  1.117643e+02  8.120796e+01
Top25perc    -61.22540945 -2.432748e+01  0.000000e+00
F.Undergrad   -0.05188654  0.000000e+00  0.000000e+00
P.Undergrad    0.03533770  0.000000e+00  0.000000e+00
Outstate       0.46090068  3.791567e-01  3.135176e-01
Room.Board     0.29042405  2.790748e-01  7.089351e-03
Books          0.69889312  2.469851e-01  0.000000e+00
Personal       0.18829646  9.332820e-02  0.000000e+00
PhD           12.76956826  7.133918e+00  0.000000e+00
Terminal      18.64905725  1.501643e+01  0.000000e+00
S.F.Ratio   -377.23545200 -3.760600e+02 -2.005979e+02
perc.alumni    7.67517822  0.000000e+00  0.000000e+00
Grad.Rate    -24.83772657 -1.282425e+01  0.000000e+00
```

The coefficients for $\lambda = 100$ and $\lambda = 1000$ illustrate a major practical difference between the lasso and ridge regression. As the value of λ increases, ridge regression shrinks the values of the coefficients toward 0, but includes all the predictors in the model. The lasso, however, not only shrinks the value of the coefficients, but sets some of them equal to 0. Thus, like forward selection and best subsets, the lasso can be considered a method of variable selection.

Choosing the Tuning Parameter with Cross-Validation

We can use cross-validation to choose a value for λ. The procedure is the same as for ridge regression. Following is 10-fold cross-validation.

We begin by setting the random seed (`set.seed(1)`) so the results will be reproducible.

```
> set.seed(1)
> k = 10
> grid = 10^seq(-2, 8, length=100)
> folds = sample(1:k, nrow(coll), replace=TRUE)
> cv.errors = matrix(NA, k, 100, dimnames=list(NULL, c(1:100)))
>
> for (j in 1:k){
>  lasso = glmnet(x[folds!=j,], y[folds!=j], alpha = 1, lambda=grid)
>   testmat = model.matrix(Expend~., data=coll[folds==j,])
>  for (i in 1:100){
>   coefi = coef(lasso)[,i]
>   pred = testmat%*%coefi
>   cv.errors[j, i] = mean((coll$Expend[folds==j] - pred)^2)
>  }
> }
>
> msep = apply(cv.errors, 2, mean)
> msepse = apply(cv.errors, 2, sd)/sqrt(k)
>
> which.min(msep)
61
> lambda.min = grid[101-60]
> min(msp)
[1] 9572940
> lambda.min
[1] 14.17474
>
> lasso.full = glmnet(x, y, alpha=1, lambda=grid)
> predict(lasso.full, s=lambda.min, type="coefficients")
18 x 1 sparse Matrix of class "dgCMatrix"
                          1
(Intercept) 7157.66366418
Private     -590.99198627
Apps           0.54573652
Accept        -0.69268226
Enroll         0.23062852
Top10perc    129.96771123
Top25perc    -57.54882111
F.Undergrad    .
P.Undergrad    0.02698833
Outstate       0.44811597
Room.Board     0.28168996
Books          0.63859288
Personal       0.17748496
PhD           11.85586219
```

```
Terminal      18.06867104
S.F.Ratio   -377.96021734
perc.alumni    6.66907068
Grad.Rate    -22.97320780
```

We now find all models whose MSEP is within one standard error of the minimum.

```
> msepsemin = which.min(msepse)
> plausibleindices = which(msep < min(msep) + msepsemin)
> names(plausibleindices) = NULL
> plausiblelambda = grid[101-plausibleindices]
> plausiblelambda
 [1] 849.75343591 705.48023107 585.70208181 486.26015801 403.70172586
 [6] 335.16026509 278.25594022 231.01297001 191.79102617 159.22827933
[11] 132.19411485 109.74987655  91.11627561  75.64633276  62.80291442
[16]  52.14008288  43.28761281  35.93813664  29.83647240  24.77076356
[21]  20.56512308  17.07352647  14.17474163  11.76811952   9.77009957
[26]   8.11130831   6.73415066   5.59081018   4.64158883   3.85352859
[31]   3.19926714   2.65608778   2.20513074   1.83073828   1.51991108
[36]   1.26185688   1.04761575   0.86974900   0.72208090   0.59948425
[41]   0.49770236   0.41320124   0.34304693   0.28480359   0.23644894
[46]   0.19630407   0.16297508   0.13530478   0.11233240   0.09326033
[51]   0.07742637   0.06428073   0.05336699   0.04430621   0.03678380
[56]   0.03053856   0.02535364   0.02104904   0.01747528   0.01450829
[61]   0.01204504   0.01000000
```

The smallest plausible value of λ is 0.01, the smallest value considered. Since this value is plausible, the ordinary least squares model ($\lambda = 0$) may predict as well as the best lasso model. On the other hand, the largest plausible of λ (849.753) will have fewer variables. We find the coefficients for this model.

```
> predict(lasso.full, s=max(plausiblelambda), type="coefficients")

(Intercept) 5589.4697439
Private        .
Apps           0.1013231
Accept         .
Enroll         .
Top10perc     91.0756120
Top25perc      .
F.Undergrad    .
P.Undergrad    .
Outstate       0.3775426
Room.Board     0.1136089
Books          .
Personal       .
```

```
PhD             .
Terminal        4.3938721
S.F.Ratio    -262.7778251
perc.alumni     .
Grad.Rate       .
```

This six-variable model is also plausible.

4.3 The Elastic Net

The elastic net is a shrinkage method that is a hybrid of ridge regression and the lasso. Recall that the penalty terms for ridge regression and the lasso are $\lambda \sum_{j=1}^{p} \beta_j^2$, and $\lambda \sum_{j=1}^{p} |\beta_j|$, respectively. The elastic net estimates minimize the quantity

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \left((1-\alpha) \sum_{j=1}^{p} \beta_j^2 + \alpha \sum_{j=1}^{p} |\beta_j| \right)$$

The tuning parameter α is a number between 0 and 1 that is chosen by the data analyst. Thus the penalty term for the elastic net is a weighted average of the penalty terms for ridge regression and the lasso. Note that $\alpha = 0$ corresponds to ridge regression and $\alpha = 1$ corresponds to the lasso.

The value of α is often chosen by cross-validation, along with the value of λ. The R command

```
glmnet(x, y, alpha=al, lambda = grid)
```

performs the elastic net using the value `al` for α for every value of λ in the array `grid`.

An Example

We use the elastic net to predict expenditures (**Expend**) using the College data set. The R code essentially consists of the code like that used for ridge regression or the lasso enclosed in a loop that runs through a selection of values for α. Following is a 10-fold cross-validation using values of α in increments of 0.1, and 100 values of λ from 10^{-2} to 10^{10}.

```
> library(glmnet)
> x = model.matrix(Expend~., data=coll)[,-1]
> y = coll$Expend

> lambda.min = rep(NA, 11)  #Initialize array for minimum lambda for each alpha
> msep.min = rep(NA, 11)    #Initialize array for minimum MSEP for each alpha
```

```
> set.seed(1)
> k = 10
> grid = 10^seq(-2, 10, length=100)
> folds = sample(1:k, nrow(coll), replace=TRUE)
> cv.errors = matrix(NA, k, 100, dimnames=list(NULL, c(1:100)))

> for (m in 1:11) {         #Loop over values of 0, 0.1, ..., 0.9, 1 for alpha
>  al = (m-1)/10

>  for (j in 1:k){
>   enet = glmnet(x[folds!=j,], y[folds!=j], alpha = al, lambda=grid)
>   testmat = model.matrix(Expend~., data=coll[folds==j,])
>   for (i in 1:100){
>    coefi = coef(enet)[,i]
>    pred = testmat%*%coefi
>    cv.errors[j, i] = mean((y[folds==j] - pred)^2)
>   }
>  }
>  msep = apply(cv.errors, 2, mean)
>  msepse = apply(cv.errors, 2, sd)/sqrt(k)
>
>  whichindex = which.min(msep)
>  minindex = 101 - whichindex
>  lambda.min[m] = grid[minindex]        #Minimum lambda for alpha = (m-1)/10
>  msep.min[m] = msep[whichindex]        #Minimum MSEP for alpha = (m-1)/10
> }

> alphaindex = which.min(msep.min)
> alpha.min = (alphaindex - 1)/10    #Value of alpha with smallest minimum MSEP
> lambda.min.min = lambda.min[alphaindex]        #Minimum lambda for that alpha
> alpha.min
[1] 0.2
> lambda.min.min
[1] 43.28761
```

```
> enet.full = glmnet(x, y, alpha=alpha.min, lambda = grid)
> predict(enet.full, s=lambda.min.min, type="coefficients")
18 x 1 sparse Matrix of class "dgCMatrix"
                        1
(Intercept) 7034.73347673
Private     -601.62609670
Apps           0.51221505
Accept        -0.63738235
Enroll         0.21011917
Top10perc    128.57573695
Top25perc    -55.72887839
F.Undergrad    .
P.Undergrad    0.02848376
Outstate       0.44185509
Room.Board     0.30083503
Books          0.67250680
Personal       0.18598058
PhD           12.38987916
Terminal      17.60697180
S.F.Ratio   -378.84111552
perc.alumni    8.21804174
Grad.Rate    -22.99956557
```

5 Vectors and Matrices

5.1 Vectors

Definition 5.1. A real vector of dimension n is an n-tuple of real numbers. For purposes of calculation, we distinguish between a row vector $(x_1, .., x_n)$ and a column vector $\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$. We will denote vectors with boldface letters; e.g. $\mathbf{x}$. By default, $\mathbf{x}$ will denote a column vector.

Whenever $\mathbf{x}$ is a (column) vector, $\mathbf{x}^T$ will represent the same vector considered as a row. If the components of $\mathbf{x}$ are $x_1, ..., x_n$, we will sometimes write $\mathbf{x} = (x_1, ..., x_n)^T$.

The set of all n-dimensional vectors is called $\mathcal{R}^n$.

Definition 5.2. Let $\mathbf{x} = (x_1, ..., x_n)^T$ be a vector, and let c be a real number. Then the product $c\mathbf{x} = (cx_1, ..., cx_n)^T$. The real number c is sometimes called a **scalar**, and the process of multiplying a vector by a scalar is called **scalar multiplication**.

Definition 5.3. Let $\mathbf{x} = (x_1, ..., x_n)^T$ and $\mathbf{y} = (y_1, ..., y_n)^T$ be vectors. The sum of $\mathbf{x}$ and $\mathbf{y}$ is $\mathbf{x} + \mathbf{y} = (x_1 + y_1, ..., x_n + y_n)^T$.

Definition 5.4. Let $\mathbf{x}_1, ..., \mathbf{x}_k$ be vectors and let $c_1, ..., c_k$ be scalars. The vector $c_1\mathbf{x}_1 + \cdots c_k\mathbf{x}_k$ is called a **linear combination** of $\mathbf{x}_1, ..., \mathbf{x}_k$.

Definition 5.5. Let $\mathbf{x}_1, ..., \mathbf{x}_k$ be vectors. The collection of all linear combinations of $\mathbf{x}_1, ..., \mathbf{x}_k$ is called the **span** of $\mathbf{x}_1, ..., \mathbf{x}_k$.

Definition 5.6. Let $\mathbf{x}_1, ..., \mathbf{x}_k$ be vectors. If there exist scalars $a_1, ..., a_k$, not all 0, such that $a_1\mathbf{x}_1 + \cdots a_k\mathbf{x}_k = \mathbf{0}$, the vectors $\mathbf{x}_1, ..., \mathbf{x}_k$ are said to be **linearly dependent**.

Example 5.1. The vectors $\mathbf{x}_1 = (2, 1, 1)^T$, $\mathbf{x}_2 = (-1, 2, -3)^T$, and $\mathbf{x}_3 = (4, 7, -3)^T$ are linearly dependent, because $3\mathbf{x}_1 + 2\mathbf{x}_2 - \mathbf{x}_3 = \mathbf{0}$.

Definition 5.7. Vectors $\mathbf{x}_1, ..., \mathbf{x}_k$ are **linearly independent** if they are not linearly dependent. Equivalently, $\mathbf{x}_1, ..., \mathbf{x}_k$ are linearly independent if $a_1\mathbf{x}_1 + \cdots a_k\mathbf{x}_k = \mathbf{0}$ only if $a_1 = a_2 = \cdots = a_k = 0$.

Example 5.2. Show that the vectors $(2, 3, 1)^T$, $(1, 0, 2)^T$, $(-1, 1, -2)^T$ are linearly independent.

Solution: Let a_1, a_2, and a_3 be constants such that $a_1(2, 3, 1)^T + a_2(1, 0, 2)^T + a_3(-1, 1, -2)^T = (0, 0, 0)^T$. This is equivalent to the following system of linear equations.

$$\begin{array}{rcrcrcl} 2a_1 & + & a_2 & - & a_3 & = & 0 \\ 3a_1 & & & + & 2a_3 & = & 0 \\ a_1 & + & 2a_2 & - & 2a_3 & = & 0 \end{array}$$

Solving, we see that the only solution is $a_1 = a_2 = a_3 = 0$.

Definition 5.8. Let $\mathbf{x}_1, ..., \mathbf{x}_n$ be n-dimensional vectors. If $\mathbf{x}_1, ..., \mathbf{x}_n$ are linearly independent and the span of $\mathbf{x}_1, ..., \mathbf{x}_n$ is $\mathcal{R}^n$, then $\mathbf{x}_1, ..., \mathbf{x}_n$ are a **basis** for $\mathcal{R}^n$.

Example 5.3. The vectors $(1, 0, 0)^T$, $(0, 1, 0)^T$, $(0, 0, 1)^T$ form a basis for $\mathcal{R}^3$. Any vector $(x, y, z)^T$ can be expressed as $x(1, 0, 0)^T + y(0, 1, 0)^T + z(0, 0, 1)^T$.

In general, let $(1, 0, 0, ..., 0)^T$, $(0, 1, 0, 0, ..., 0)^T$, ..., $(0, 0, ..., 0, 1)^T$ be vectors in $\mathcal{R}^n$ where the ith vector has a 1 in the ith position and 0s elsewhere. This is a basis for $\mathcal{R}^n$, called the **canonical basis**.

Theorem 5.1. Any set of n linearly independent vectors is a basis for $\mathcal{R}^n$.

Proof: Omitted.

Example 5.4. The vectors $(2, 3, 1)$, $(1, 0, 2)$, $(-1, 1, -2)$ are a basis for $\mathcal{R}^3$ because there are three of them and they are linearly independent.

Theorem 5.2. Let $\mathbf{x}_1, ..., \mathbf{x}_n$ be a basis for $\mathcal{R}^n$. Then any vector in $\mathcal{R}^n$ can be expressed uniquely as a linear combination of $\mathbf{x}_1, ..., \mathbf{x}_n$.

Proof: Let $\mathbf{v} \in \mathcal{R}^n$. Now $\mathbf{x}_1, ..., \mathbf{x}_n$ span $\mathcal{R}^n$ because they are a basis. Therefore there exist coefficients $a_1, ..., a_n$ such that $a_1\mathbf{x}_1 + \cdots a_n\mathbf{x}_n = \mathbf{v}$. We must show that $a_1, ..., a_n$ are the only such coefficients.

Let $b_1, ..., b_n$ be coefficients such that $b_1\mathbf{x}_1 + \cdots b_n\mathbf{x}_n = \mathbf{v}$. Then $a_1\mathbf{x}_1 + \cdots + a_n\mathbf{x}_n = b_1\mathbf{x}_1 + \cdots + b_n\mathbf{x}_n$. Subtracting, we have $(a_1 - b_1)\mathbf{x}_1 + \cdots (a_n - b_n)\mathbf{x}_n = \mathbf{0}$. But since $\mathbf{x}_1, ..., \mathbf{x}_n$ are linearly independent, this implies that $a_1 - b_1 = \cdots = a_n - b_n = 0$, or equivalently $a_1 = b_1$, $a_2 = b_2$, ..., $a_n = b_n$.

Definition 5.9. Let $\mathbf{x} = (x_1, ..., x_n)^T \in \mathcal{R}^n$. The **length** of $\mathbf{x}$ is $|\mathbf{x}| = \sqrt{x_1^2 + \cdots x_n^2}$.

Definition 5.10. Let $\mathbf{x} = (x_1, ..., x_n)^T$ and $\mathbf{y} = (y_1, ..., y_n)^T$ be vectors in $\mathcal{R}^n$. The **inner product** of $\mathbf{x}$ and $\mathbf{y}$ is $\mathbf{x} \cdot \mathbf{y} = x_1y_1 + x_2y_2 + \cdots + x_ny_n$.

Remark 5.1. The inner product is sometimes called the dot product.

Theorem 5.3. Let $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{u}$ be vectors in $\mathcal{R}^n$. Let a be a constant. Then
1) $\mathbf{x} \cdot \mathbf{x} = |\mathbf{x}|^2$
2) $\mathbf{x} \cdot (a\mathbf{u}) = a(\mathbf{x} \cdot \mathbf{u})$
3) $(\mathbf{x} + \mathbf{y}) \cdot \mathbf{u} = \mathbf{x} \cdot \mathbf{u} + \mathbf{y} \cdot \mathbf{u}$

Proof:
1) $\mathbf{x} \cdot \mathbf{x} = x_1^2 + \cdots + x_n^2 = |\mathbf{x}|^2$
2) $\mathbf{x} \cdot (a\mathbf{u}) = x_1(au_1) + \cdots + x_n(au_n) = a(x_1u_1 + \cdots + x_nu_n) = a(\mathbf{x} \cdot \mathbf{u})$
3) $(\mathbf{x} + \mathbf{y}) \cdot \mathbf{u} = (x_1 + y_1)u_1 + \cdots + (x_n + y_n)u_n = x_1u_1 + y_1u_1 + \cdots + x_nu_n + y_nu_n = x_1u_1 + \cdots + x_nu_n + y_1u_1 + \cdots + y_nu_n = \mathbf{x} \cdot \mathbf{u} + \mathbf{y} \cdot \mathbf{u}$

Definition 5.11. Let $\mathbf{x}$ and $\mathbf{y}$ be vectors in $\mathcal{R}^n$. $\mathbf{x}$ and $\mathbf{y}$ are **orthogonal** if $\mathbf{x} \cdot \mathbf{y} = 0$. We write $\mathbf{x} \perp \mathbf{y}$. We say that that vectors $\mathbf{x}_1, ..., \mathbf{x}_n$ are orthogonal if $\mathbf{x}_i \perp \mathbf{x}_j$ for every $i \neq j$.

Theorem 5.4.
1) $\mathbf{0}$ is orthogonal to every vector.
2) Let $\mathbf{x}_1, ..., \mathbf{x}_k \in \mathcal{R}^n$. If $\mathbf{z} \in \mathcal{R}^n$ is orthogonal to each $\mathbf{x}_i$, then $\mathbf{z}$ is orthogonal to every vector in the span of $\mathbf{x}_1, ..., \mathbf{x}_k$.
3) Let $\mathbf{x}_1, ..., \mathbf{x}_k$ be orthogonal, with $\mathbf{x}_i \neq \mathbf{0}$ for all i. Then $\mathbf{x}_1, ..., \mathbf{x}_k$ are linearly independent.

Proof:
1) $\mathbf{x} \cdot \mathbf{0} = 0$ for any vector $\mathbf{x}$.
2) Let $\mathbf{v} = a_1\mathbf{x}_1 + \cdots + a_k\mathbf{x}_k$ be a linear combination of $\mathbf{x}_1, ..., \mathbf{x}_k$. Since $\mathbf{z} \perp \mathbf{x}_i$ for all i, $\mathbf{z} \cdot \mathbf{x}_i = 0$ for all i. Then $\mathbf{z} \cdot \mathbf{v} = \sum_{i=1}^{k} a_i \mathbf{z}_i \cdot \mathbf{x}_i = 0$.
3) Let $\mathbf{x}_1, ..., \mathbf{x}_k$ be orthogonal. Let $a_1, ..., a_k$ be constants such that $a_1\mathbf{x}_1 + \cdots + a_k\mathbf{x}_k = \mathbf{0}$. Let a_i be any coefficient. We show that $a_i = 0$.
$0 = a_1\mathbf{x}_1 \cdot \mathbf{x}_i + \cdots + a_i\mathbf{x}_i \cdot \mathbf{x}_i + \cdots + a_k\mathbf{x}_k \cdot \mathbf{x}_i = a_i(0) + \cdots + a_i\mathbf{x}_i \cdot \mathbf{x}_i + \cdots + a_k(0) = a_i|\mathbf{x}_i|^2 = 0$.
Since $\mathbf{x}_i \neq \mathbf{0}$, $|\mathbf{x}_i|^2 \neq 0$. Therefore $a_i = 0$.

Theorem 5.5. (Pythagoras) Let $\mathbf{x}$, and $\mathbf{y}$ be vectors in $\mathcal{R}^n$. Then $|\mathbf{x} + \mathbf{y}|^2 = |\mathbf{x}|^2 + |\mathbf{y}|^2$ if and only if $\mathbf{x} \perp \mathbf{y}$.

Proof:
$|\mathbf{x} + \mathbf{y}|^2 = \sum_{i=1}^{n} x_i^2 + y_i^2 + 2x_iy_i = \sum_{i=1}^{n} x_i^2 + \sum_{i=1}^{n} y_i^2 + 2\sum_{i=1}^{n} x_iy_i = |\mathbf{x}|^2 + |\mathbf{y}|^2 + 2\mathbf{x} \cdot \mathbf{y}$
Therefore $|\mathbf{x} + \mathbf{y}|^2 = |\mathbf{x}|^2 + |\mathbf{y}|^2$ if and only if $\mathbf{x} \cdot \mathbf{y} = 0$.

5.2 Basic Facts About Matrices

A matrix can be thought of as a rectangular array of numbers. A matrix with n rows and k columns is an $n \times k$ matrix.

Matrix addition:
If $\mathbf{A}$ and $\mathbf{B}$ have the same dimensions, then $\mathbf{A} + \mathbf{B}$ is computed by adding elementwise.

Matrix multiplication:
Let $\mathbf{A}$ and $\mathbf{B}$ be matrices. If the number of columns of $\mathbf{A}$ is the same as the number of rows of $\mathbf{B}$, the product $\mathbf{AB}$ exists. Specifically, if $\mathbf{A}$ is $n \times k$ and $\mathbf{B}$ is $k \times p$, the product $\mathbf{AB}$ is an $n \times p$ matrix. The element in the ith row and jth column of $\mathbf{AB}$ is the dot product of the ith row of $\mathbf{A}$ with the jth column of $\mathbf{B}$.

Example 5.5.

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix} \begin{pmatrix} 3 & 7 \\ 4 & 8 \\ 5 & 9 \\ 6 & 10 \end{pmatrix} = \begin{pmatrix} 50 & 90 \\ 122 & 226 \end{pmatrix}$$

Note the following:

- If $\mathbf{A}$ is $n \times k$ and $\mathbf{B}$ is $k \times p$, then $\mathbf{AB}$ is $n \times p$.
- $\mathbf{AB}$ and $\mathbf{BA}$ both exist if and only if the number of rows of $\mathbf{A}$ is equal to the number of columns of $\mathbf{B}$ and the number of columns of $\mathbf{A}$ is equal to the number of rows of $\mathbf{B}$. If $\mathbf{A}$ is $n \times k$ and $\mathbf{B}$ is $k \times n$, then $\mathbf{AB}$ is $n \times n$ and and $\mathbf{BA}$ is $k \times k$.
- Matrix multiplication is not commutative. In general, $\mathbf{AB} \neq \mathbf{BA}$.

Definition 5.12. Let $\mathbf{A}$ be a matrix. The transpose of $\mathbf{A}$, denoted $\mathbf{A}^T$, is the matrix formed by interchanging the rows and columns of $\mathbf{A}$.

Example 5.6. Let $\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix}$. Then $\mathbf{A}^T = \begin{pmatrix} 1 & 5 \\ 2 & 6 \\ 3 & 7 \\ 4 & 8 \end{pmatrix}$.

Proposition 5.1.
1) If $\mathbf{A}$ is $n \times k$, then $\mathbf{A}^T$ is $k \times n$, $\mathbf{A}\mathbf{A}^T$ is $n \times n$, and $\mathbf{A}^T\mathbf{A} = k \times k$.
2) A column vector in $\mathcal{R}^n$ is a $n \times 1$ matrix, and a row vector is a $1 \times n$ matrix.
3) Let $\mathbf{u}$ and $\mathbf{v}$ be vectors in $\mathcal{R}^n$. Then $\mathbf{u} \cdot \mathbf{v} = \mathbf{u}^T\mathbf{v} = \mathbf{v}^T\mathbf{u}$.
4) Let $\mathbf{u}$ be a vector in $\mathcal{R}^n$. Then $\mathbf{u} \cdot \mathbf{u} = \mathbf{u}^T\mathbf{u} = \sum_{i=1}^n u_i^2 = |\mathbf{u}|^2$.

Proposition 5.2.
1) Let $\mathbf{A}$ be $n \times k$, and let $\mathbf{v}$ be a $k \times 1$ column vector. Then $\mathbf{Av}$ is a $n \times 1$ column vector.
2) Let $\mathbf{A}$ be $n \times k$, and let $\mathbf{v}$ be a $n \times 1$ column vector. Then $\mathbf{v}^T\mathbf{A}$ is a $1 \times k$ row vector.
3) Let $\mathbf{A}$ be $n \times k$, let $\mathbf{v}$ be $n \times 1$, and let $\mathbf{u}$ be $k \times 1$. Then $\mathbf{u}^T\mathbf{Av}$ is a real number (1×1).

Proposition 5.3. For any matrices $\mathbf{A}$ and $\mathbf{B}$ for which $\mathbf{AB}$ exists, $(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T$.

Definition 5.13. The $n \times n$ identity matrix is the $n \times n$ matrix whose main diagonal elements are 1, and all of whose other elements are 0. An identity matrix is generally denoted $\mathbf{I}$.

Proposition 5.4. For any $n \times n$ matrix $\mathbf{A}$, $\mathbf{AI} = \mathbf{IA} = \mathbf{A}$.

Definition 5.14. A matrix is **square** if the number of rows is the same as the number of columns.

Definition 5.15. A matrix $\mathbf{A}$ is symmetric if $\mathbf{A}^T = \mathbf{A}$. Note that a symmetric matrix must be square.

5.3 Determinants

Definition 5.16. A **permutation** of a set of objects is an ordering of them. For example, there are 6 permutations of the integers 1,2,3: 123, 132, 213, 231, 312, 321.

Definition 5.17. A **transposition** is a permutation that involves interchanging two objects. For example, 1432 is a transposition of 1234.

Theorem 5.6. Any permutation can be obtained through a sequence of transpositions. For any permutation, there are infinitely many sequences that produce it. However, it will be the case either that all the sequences have an even number of transpositions or that all the sequences have an odd number of transpositions.

Proof: Omitted.

Definition 5.18. A permutation produced by an even number of transpositions is called an **even permutation**, and a permutation produced by an odd number of transpositions is called an **odd permutation**.

Example 5.7. The permutation 2431 is an even permutation of 1234, because it is obtainable with two transpositions: $1234 \to 2134 \to 2431$. While there are other sequences of transpositions that produce this permutation, they all have an even number of transpositions.

The permutation 3421 is an odd permutation of 1234, because it is obtainable with three transpositions: $1234 \to 3214 \to 3412 \to 3421$. While there are other sequences of transpositions that produce this permutation, they all have an odd number of transpositions.

Definition 5.19. Let $\mathbf{A}$ be an $n \times n$ square matrix. Let $j_1, ..., j_n$ be a permutation of the integers $1, 2, ..., n$. The **determinant** of $\mathbf{A}$ is

$$|\mathbf{A}| = \sum (-1)^p a_{1,j_1}\, a_{2,j_2} \cdots a_{n,j_n}$$

where the sum is taken of all permutations $j_1, j_2, ..., j_n$ of $1, 2, ..., n$; $p = 1$ if the permutation is odd and $p = 2$ if the permutation is even.

Example 5.8. Let $\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ be a 2×2 matrix. The determinant of $\mathbf{A}$ is $|\mathbf{A}| = a_{11}a_{22} - a_{12}a_{21}$.

In practice, determinants are calculated electronically. Here is an example using R:

Example 5.9. Find the determinant of $\begin{bmatrix} 1.55 & -0.09 & -0.05 & -1.71 & -0.09 & -1.31 \\ 0.42 & -0.03 & -0.83 & 1.70 & 0.37 & 0.51 \\ 0.06 & 1.16 & -1.44 & 0.41 & 0.49 & 1.00 \\ 0.55 & 0.09 & 0.37 & 1.06 & 0.72 & -0.72 \\ 1.17 & -1.35 & -1.08 & 0.11 & -0.59 & 0.12 \\ 1.40 & -0.95 & 0.17 & 1.07 & 0.65 & 1.05 \end{bmatrix}$.

Solution: We use R:

```
#First we enter the data as a string:

> mat = c(1.55, 0.42, 0.06, 0.55, -1.17, 1.40, -0.09, -0.03, 1.16, 0.09, -1.35,
-0.95, -0.05, -0.83, -1.44, 0.37, -1.08  0.17, -1.71, 1.70, 0.41, 1.06, 0.11,
1.07, -0.09, 0.37, 0.49, 0.72, -0.59, 0.65, -1.31, 0.51, 1.00, -0.72, 0.12, 1.05)

#Then convert it to a matrix.

> mat = matrix(mat, 6, 6)
> mat
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]
```

```
[1,]  1.55 -0.09 -0.05 -1.71 -0.09 -1.31
[2,]  0.42 -0.03 -0.83  1.70  0.37  0.51
[3,]  0.06  1.16 -1.44  0.41  0.49  1.00
[4,]  0.55  0.09  0.37  1.06  0.72 -0.72
[5,] -1.17 -1.35 -1.08  0.11 -0.59  0.12
[6,]  1.40 -0.95  0.17  1.07  0.65  1.05

#Now compute the determinant.

> det(mat)
[1] 14.88555

#Just for fun, we will compute the determinant of the transpose. It's the
#same as the determinant of the original matrix.

> det(t(mat))
[1] 14.88555
```

Proposition 5.5. Let $\mathbf{A}$ be a square matrix. Then $|\mathbf{A}^T| = |\mathbf{A}|$.

5.4 Singular and Nonsingular Matrices

Definition 5.20. The **rank** of a matrix is the maximum number of columns that can be chosen to be linearly independent. Equivalently, the rank can be defined as the maximum number of independent rows.

Definition 5.21. A matrix has full column rank if its columns are linearly independent, in other words, if its rank is equal to the number of columns.

Note that a matrix cannot have full column rank unless the number of columns is less than or equal to the number of rows.

Theorem 5.7. Let $\mathbf{A}$ be an $n \times p$ matrix of full column rank and let $\mathbf{x}$ be a $p \times 1$ vector.
1) Then $\mathbf{Ax} \neq \mathbf{0}$ unless $\mathbf{x} = \mathbf{0}$.
2) Conversely, if $\mathbf{A}$ is an $n \times p$ matrix not of full rank, there will exist a vector $\mathbf{x} \neq \mathbf{0}$ for which $\mathbf{Ax} = \mathbf{0}$.

Proof:
1) This follows because the columns of $\mathbf{A}$ are linearly independent, and $\mathbf{Ax}$ is a linear combination of the columns of $\mathbf{A}$.

2) This follows because the columns of $\mathbf{A}$ are linearly dependent.

Definition 5.22. Let $\mathbf{A}$ be a $n \times n$ square matrix. If there exists a matrix $\mathbf{A}^{-1}$ such that $\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$, $\mathbf{A}^{-1}$ is called the **inverse** of $\mathbf{A}$.

Definition 5.23. A matrix whose inverse exists is said to be nonsingular.

Proposition 5.6. For any nonsingular matrices $\mathbf{A}$ and $\mathbf{B}$, $(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$. To see this, note that $(\mathbf{AB})(\mathbf{B}^{-1}\mathbf{A}^{-1}) = \mathbf{A}(\mathbf{B}\mathbf{B}^{-1})\mathbf{A}^{-1} = \mathbf{A}(\mathbf{I})\mathbf{A}^{-1} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$.

Proposition 5.7. Let $\mathbf{A}$ be a nonsingular matrix. Then $\mathbf{A}^T$ is nonsingular, and $(\mathbf{A}^T)^{-1} = (\mathbf{A}^{-1})^T$. To see this, note that $(\mathbf{A}^{-1})^T\mathbf{A}^T = (\mathbf{A}\mathbf{A}^{-1})^T = \mathbf{I}$.

Theorem 5.8. Let $\mathbf{A}$ and $\mathbf{B}$ be $n \times n$ square matrices. Then
1) $|\mathbf{A}| = |\mathbf{A}^T|$
2) $|\mathbf{AB}| = |\mathbf{A}||\mathbf{B}|$
3) Let c be a scalar. Then $|c\mathbf{A}| = c^n|\mathbf{A}|$.
4) If $\mathbf{A}$ is nonsingular, then $|\mathbf{A}^{-1}| = 1/|\mathbf{A}|$.

Proof:
1) is straightforward but tedious. The proof of 2) is omitted. 3) is straightforward. 4) follows from 2) by noting that $|\mathbf{A}||\mathbf{A}^{-1}| = |\mathbf{A}\mathbf{A}^{-1}| = |\mathbf{I}| = 1$.

Theorem 5.9. Let $\mathbf{A}$ be a square matrix. The following statements are equivalent.
1) The columns of $\mathbf{A}$ are linearly independent.
2) $\mathbf{Ax} \neq \mathbf{0}$ unless $\mathbf{x} = \mathbf{0}$.
3) $\mathbf{A}$ is nonsingular.
4) $|\mathbf{A}| \neq 0$.

Proof:
1) $\Rightarrow$ 2): $\mathbf{Ax}$ is a linear combination of the columns of $\mathbf{A}$ whose coefficients are the elements of $\mathbf{x}$. Since the columns of $\mathbf{A}$ are linearly independent, $\mathbf{Ax} \neq \mathbf{0}$ unless $\mathbf{x} = \mathbf{0}$.

2) $\Rightarrow$ 3): Let $\mathbf{v}_1, ..., \mathbf{v}_n$ be a basis for $\mathcal{R}^n$. For $i = 1, ..., n$, Let $\mathbf{u}_i = \mathbf{A}\mathbf{v}_i$. We show that $\mathbf{u}_1, ..., \mathbf{u}_n$ is a basis for $\mathcal{R}^n$. Let $a_1, ..., a_n$ be real numbers such that $a_1\mathbf{u}_1 + \cdots + a_n\mathbf{u}_n = \mathbf{0}$. Then $a_1\mathbf{A}\mathbf{v}_1 + \cdots + a_n\mathbf{A}\mathbf{v}_n = \mathbf{A}(a_1\mathbf{v}_1 + \cdots + a_n\mathbf{v}_n) = \mathbf{0}$. It follows from 2) that $a_1\mathbf{v}_1 + \cdots +$

$a_n\mathbf{v}_n = \mathbf{0}$. Since $\mathbf{v}_1, ..., \mathbf{v}_n$ is a basis for $\mathcal{R}^n$, $a_1 = \cdots = a_n = 0$. It follows that $\mathbf{u}_1, ..., \mathbf{u}_n$ is a basis for $\mathcal{R}^n$.

Now let $\mathbf{e}_i$ be the vector with a 1 in the ith position and zeros elsewhere, and let $\alpha_1, ..., \alpha_n$ be real numbers such that $\alpha_1\mathbf{u}_1 + \cdots + \alpha_n\mathbf{u}_n = \mathbf{e}_i$. Let $\mathbf{w}_i = \alpha_1\mathbf{v}_1 + \cdots + \alpha_n\mathbf{v}_n$. Then $\mathbf{A}\mathbf{w}_i = \mathbf{A}(\alpha_1\mathbf{v}_1 + \cdots + \alpha_n\mathbf{v}_n) = \alpha_1\mathbf{A}\mathbf{v}_1 + \cdots + \alpha_n\mathbf{A}\mathbf{v}_n = \alpha_1\mathbf{u}_1 + \cdots + \alpha_n\mathbf{u}_n = \mathbf{e}_i$.

Let $\mathbf{W}$ be the $n \times n$ matrix whose ith column is $\mathbf{w}_i$. Then $\mathbf{A}\mathbf{W} = \mathbf{I}$, so $\mathbf{W} = \mathbf{A}^{-1}$.

3) $\Rightarrow$ 4): Let $\mathbf{I}$ be the $n \times n$ identity matrix. Then $1 = |\mathbf{I}| = |\mathbf{A}\mathbf{A}^{-1}| = |\mathbf{A}||\mathbf{A}^{-1}|$, so $|\mathbf{A}| \neq 0$.

4) $\Rightarrow$ 1): Omitted.

The following theorem follows from Theorem 5.9.

Theorem 5.10. Let $\mathbf{A}$ be a square matrix. The following statements are equivalent.
1) The columns of $\mathbf{A}$ are linearly dependent.
2) $\mathbf{A}\mathbf{x} = \mathbf{0}$ for some $\mathbf{x} \neq \mathbf{0}$.
3) $\mathbf{A}$ is singular, that is, $\mathbf{A}^{-1}$ does not exist.
4) $|\mathbf{A}| = 0$.

Example 5.10. Let $\mathbf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ be a 2×2 matrix. If $ad - bc \neq 0$, then $\mathbf{A}$ is nonsingular, and $\mathbf{A}^{-1} = \begin{pmatrix} \frac{d}{ad-bc} & \frac{-b}{ad-bc} \\ \frac{-c}{ad-bc} & \frac{a}{ad-bc} \end{pmatrix}$

Definition 5.24. Let $\mathbf{A}$ be an $n \times n$ square matrix. The **trace** of $\mathbf{A}$ is the sum of the diagonal elements of $\mathbf{A}$; $tr(\mathbf{A}) = \sum_{i=1}^{n} a_{ii}$.

Theorem 5.11. Let $\mathbf{A}$ be $n \times p$ and $\mathbf{B}$ be $p \times n$. Then $tr(\mathbf{A}\mathbf{B}) = tr(\mathbf{B}\mathbf{A})$.

Proof: The ith diagonal element of $\mathbf{A}\mathbf{B}$ is $\sum_{j=1}^{p} a_{ij}b_{ji}$.
Thus $tr(\mathbf{A}\mathbf{B}) = \sum_{i=1}^{n}\sum_{j=1}^{p} a_{ij}b_{ji} = \sum_{i=1}^{p}\sum_{j=1}^{n} b_{ij}a_{ji} = tr(\mathbf{B}\mathbf{A})$.

5.5 Orthogonal and Orthonormal Matrices

Definition 5.25. A square matrix $\mathbf{A}$ is said to be **orthogonal** if the columns of $\mathbf{A}$ are orthogonal. $\mathbf{A}$ is **orthonormal** if the columns of $\mathbf{A}$ are orthonormal.

Theorem 5.12. Let $\mathbf{A}$ be a square matrix. Then $\mathbf{A}^T\mathbf{A} = \mathbf{I}$ if and only if $\mathbf{A}$ is orthonormal.

Proof:
Let $\mathbf{a}_1, ..., \mathbf{a}_n$ be the columns of $\mathbf{A}$. The ij element of $\mathbf{A}^T\mathbf{A}$ is $\mathbf{a}_i^T\mathbf{a}_j$. Therefore $\mathbf{A}^T\mathbf{A} = \mathbf{I}$ if and only if $\mathbf{a}_i^T\mathbf{a}_j = 0$ if $i \neq j$ and 1 if $i = j$. This occurs if and only if $\mathbf{a}_1, ...\mathbf{a}_n$ are orthonormal.

Theorem 5.13. Let $\mathbf{A}$ be a square matrix. If $\mathbf{A}$ is orthonormal then $\mathbf{A}^T$ is orthonormal and $\mathbf{A}^T = \mathbf{A}^{-1}$.

Proof:
We know from Theorem 5.12 that $\mathbf{A}^T\mathbf{A} = \mathbf{I}$. We must show that $\mathbf{A}\mathbf{A}^T = \mathbf{I}$. It will follow that $\mathbf{A}^T = \mathbf{A}^{-1}$ and from Theorem 5.12 that $\mathbf{A}^T$ is orthonormal.

Let $\mathbf{A}$ be orthonormal. Since the columns of $\mathbf{A}$ are orthogonal, they are linearly independent. Since the columns of $\mathbf{A}$ are linearly independent. the rows are linearly independent as well. Therefore the columns of $\mathbf{A}^T$ are linearly independent.

Since $\mathbf{A}^T\mathbf{A} = \mathbf{I}$, $\mathbf{A}^T = (\mathbf{A}^T\mathbf{A})\mathbf{A}^T = \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)$. It follows that $\mathbf{A}^T(\mathbf{I} - \mathbf{A}\mathbf{A}^T) = \mathbf{0}$. Since the columns of $\mathbf{A}^T$ are linearly independent, $\mathbf{I} - \mathbf{A}\mathbf{A}^T = \mathbf{0}$, so $\mathbf{A}\mathbf{A}^T = \mathbf{I}$.

5.6 Eigenvectors and Eigenvalues

Definition 5.26. Let $\mathbf{A}$ be an $n \times n$ square matrix. A vector $\mathbf{e} \neq \mathbf{0}$ is called an **eigenvector** of $\mathbf{A}$ if $\mathbf{A}\mathbf{e} = \lambda\mathbf{e}$ for some constant $\lambda \neq 0$. The constant λ is called the **eigenvalue** for the eigenvector $\mathbf{e}$.

Example 5.11. Show that $\begin{bmatrix} 3 \\ 2 \\ 6 \end{bmatrix}$ is an eigenvector of $\begin{bmatrix} 9 & 9 & -4 \\ -14 & 28 & 0 \\ -6 & 15 & 5 \end{bmatrix}$. Find the corresponding eigenvalue.

Solution:

$$\begin{bmatrix} 9 & 9 & -4 \\ -14 & 28 & 0 \\ -6 & 15 & 5 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \\ 6 \end{bmatrix} = \begin{bmatrix} 21 \\ 14 \\ 42 \end{bmatrix} = 7 \begin{bmatrix} 3 \\ 2 \\ 6 \end{bmatrix}.$$

Since $\begin{bmatrix} 9 & 9 & -4 \\ -14 & 28 & 0 \\ -6 & 15 & 5 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \\ 6 \end{bmatrix}$ is a multiple of $\begin{bmatrix} 3 \\ 2 \\ 6 \end{bmatrix}$, $\begin{bmatrix} 3 \\ 2 \\ 6 \end{bmatrix}$ is an eigenvector of $\begin{bmatrix} 9 & 9 & -4 \\ -14 & 28 & 0 \\ -6 & 15 & 5 \end{bmatrix}$.
Its eigenvalue is 7.

Definition 5.27. Let $\mathbf{A}$ be an $n \times n$ square matrix. Let $\mathbf{I}$ be the $n \times n$ identity matrix. The function

$$q(\lambda) = |\mathbf{A} - \lambda \mathbf{I}|$$

is called the **characteristic polynomial** of $\mathbf{A}$.

Let a_{ij} denote the ijth element of $\mathbf{A}$. Note that the ijth element of the matrix $\mathbf{A} - \lambda \mathbf{I}$ is a_{ij} if $i \neq j$, and the iith element is $a_{ii} - \lambda$.

Theorem 5.14. A number λ is an eigenvalue of $\mathbf{A}$ if and only if it is a root of the characteristic polynomial, that is, if and only if $q(\lambda) = |\mathbf{A} - \lambda \mathbf{I}| = 0$.

Proof:
Let λ be a root of $q(\lambda)$. Then $|\mathbf{A} - \lambda \mathbf{I}| = 0$, so the columns of $\mathbf{A} - \lambda \mathbf{I}$ are linearly dependent. Therefore there is a nonzero vector $\mathbf{e}$ such that $(\mathbf{A} - \lambda \mathbf{I})\mathbf{e} = \mathbf{0}$. But then $\mathbf{A}\mathbf{e} = \lambda \mathbf{I}\mathbf{e} = \lambda \mathbf{e}$, so λ is an eigenvalue of $\mathbf{A}$.

Now assume that λ is an eigenvalue of $\mathbf{A}$. Let $\mathbf{e} \neq \mathbf{0}$ be an eigenvector for λ. Then $\mathbf{A}\mathbf{e} = \lambda \mathbf{e}$, so $(\mathbf{A} - \lambda \mathbf{I})\mathbf{e} = \mathbf{0}$. Since $\mathbf{e} \neq \mathbf{0}$, this implies that columns of $\mathbf{A} - \lambda \mathbf{I}$ are linearly dependent. Therefore $|\mathbf{A} - \lambda \mathbf{I}| = 0$.

Theorem 5.15. Let $\mathbf{A}$ be an $n \times n$ square matrix. Then its characteristic polynomial $q(\lambda)$ is a polynomial of degree n in λ, and the coefficient of λ^n is $(-1)^n$.

Proof:
The term in the determinant $|\mathbf{A} - \lambda\mathbf{I}|$ containing the largest power of λ is the product of the main diagonal $\prod_{i=1}^{n}(a_{ii} - \lambda)$. The largest power of λ is therefore $(-\lambda)^n = (-1)^n\lambda^n$.

It follows from Theorem 5.15 that $q(\lambda)$ can be written in the form $q(\lambda) = (-1)^n(\lambda - \lambda_1)\cdots(\lambda - \lambda_n)$ for some constants $\lambda_1, ..., \lambda_n$. The values λ_i are the roots of $q(\lambda)$ and thus the eigenvalues of $\mathbf{A}$. Some or all of the λ_i may be complex. Also, some may be equal to each other. If there are k roots equal to λ_i, then λ_i is said to be an eigenvalue of multiplicity k.

Example 5.12. Find the eigenvalues of $\begin{bmatrix} 7 & -3 \\ 6 & -2 \end{bmatrix}$.

Solution:
The characteristic polynomial is $\begin{vmatrix} 7-\lambda & -3 \\ 6 & -2-\lambda \end{vmatrix} = (7-\lambda)(-2-\lambda) - (-18) = \lambda^2 - 5\lambda + 4$.
The eigenvalues are $\lambda_1 = 1$, $\lambda_2 = 4$.

Example 5.13. Find an eigenvector of $\begin{bmatrix} 7 & -3 \\ 6 & -2 \end{bmatrix}$ for each eigenvalue.

Solution:
From Example 5.12 we know that the eigenvalues are $\lambda_1 = 1$, $\lambda_2 = 4$. We find the eigenvectors by solving systems of linear equations. The eigenvector for $\lambda_1 = 1$ is the solution to

$$\begin{bmatrix} 7 & -3 \\ 6 & -2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 1 \begin{bmatrix} x \\ y \end{bmatrix}$$

Any vector of the form $(x, 2x)$ is a solution, and thus an eigenvector with eigenvalue 1.

The eigenvector for $\lambda_2 = 4$ is the solution to

$$\begin{bmatrix} 7 & -3 \\ 6 & -2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 4 \begin{bmatrix} x \\ y \end{bmatrix}$$

Any vector of the form (x, x) is a solution, and thus an eigenvector with eigenvalue 4.

In general, if $\mathbf{Ae} = \lambda\mathbf{e}$, and c is any constant, then $\mathbf{A}(c\mathbf{e}) = c\mathbf{Ae} = c\lambda\mathbf{e} = \lambda(c\mathbf{e})$. Thus if $\mathbf{e}$ is an eigenvector of $\mathbf{A}$ with eigenvalue λ, then any multiple of $\mathbf{e}$ is also an eigenvector with eigenvalue λ.

Definition 5.28. Let $\mathbf{e}$ be an eigenvector with eigenvalue λ. The **standardized eigenvector** for the eigenvalue λ is $\mathbf{e}/|\mathbf{e}|$, which is the multiple of $\mathbf{e}$ that has length 1.

Example 5.14. Find the standardized eigenvectors of the matrix $\begin{bmatrix} 7 & -3 \\ 6 & -2 \end{bmatrix}$.

Solution:
From Example 5.13, we know that any vector of the form $(x, 2x)^T$ is an eigenvector with eigenvalue 1. The standardized eigenvector is $(x, 2x)^T/\sqrt{x^2 + (2x)^2} = (1/\sqrt{5}, 2/\sqrt{5})^T$.

Similarly, we know from Example 5.13 that any vector of the form $(x, x)^T$ is an eigenvector with eigenvalue 4. The standardized eigenvector is $(x, x)^T/\sqrt{x^2 + x^2} = (1/\sqrt{2}, 1/\sqrt{2})^T$.

In general, eigenvalues and eigenvectors are computed with software. Here is an example using R:

Example 5.15. Find the eigenvalues and eigenvectors of the matrix

$$\begin{bmatrix} 1.55 & -0.09 & -0.05 & -1.71 & -0.09 & -1.31 \\ 0.42 & -0.03 & -0.83 & 1.70 & 0.37 & 0.51 \\ 0.06 & 1.16 & -1.44 & 0.41 & 0.49 & 1.00 \\ 0.55 & 0.09 & 0.37 & 1.06 & 0.72 & -0.72 \\ 1.17 & -1.35 & -1.08 & 0.11 & -0.59 & 0.12 \\ 1.40 & -0.95 & 0.17 & 1.07 & 0.65 & 1.05 \end{bmatrix}.$$

Solution: We use R:

```
#First we enter the data as a string:

> mat = c(1.55, 0.42, 0.06, 0.55, -1.17, 1.40, -0.09, -0.03, 1.16, 0.09, -1.35,
-0.95, -0.05, -0.83, -1.44, 0.37, -1.08  0.17, -1.71, 1.70, 0.41, 1.06, 0.11,
1.07, -0.09, 0.37, 0.49, 0.72, -0.59, 0.65, -1.31, 0.51, 1.00, -0.72, 0.12, 1.05)

#Then convert it to a matrix.

> mat = matrix(mat, 6, 6)
> mat
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]
[1,]  1.55 -0.09 -0.05 -1.71 -0.09 -1.31
[2,]  0.42 -0.03 -0.83  1.70  0.37  0.51
[3,]  0.06  1.16 -1.44  0.41  0.49  1.00
[4,]  0.55  0.09  0.37  1.06  0.72 -0.72
[5,] -1.17 -1.35 -1.08  0.11 -0.59  0.12
[6,]  1.40 -0.95  0.17  1.07  0.65  1.05
```

```
#Now compute the eigenvalues and eigenvectors.

> eigen(mat)
$values
[1]  0.685277+1.978592i  0.685277-1.978592i  1.623504+0.000000i
[4] -1.114523+1.123532i -1.114523-1.123532i  0.834987+0.000000i

$vectors
                      [,1]                   [,2]           [,3]
[1,]  0.0679653+0.5128781i  0.0679653-0.5128781i -0.54567522+0i
[2,]  0.3788357-0.0274673i  0.3788357+0.0274673i  0.22206346+0i
[3,]  0.2671614-0.1690424i  0.2671614+0.1690424i -0.04002503+0i
[4,]  0.2159535+0.1765727i  0.2159535-0.1765727i  0.48348081+0i
[5,] -0.3114067+0.2000734i -0.3114067-0.2000734i  0.16264576+0i
[6,]  0.5227791+0.0000000i  0.5227791+0.0000000i -0.62539350+0i
                        [,4]                   [,5]           [,6]
[1,]  0.07370966+0.03021084i  0.07370966-0.03021084i -0.36974965+0i
[2,] -0.28911406+0.12999404i -0.28911406-0.12999404i -0.41404959+0i
[3,] -0.05311488+0.55603188i -0.05311488-0.55603188i -0.02417304+0i
[4,]  0.22686296-0.00068835i  0.22686296+0.00068835i -0.35957163+0i
[5,] -0.72552599+0.00000000i -0.72552599+0.00000000i  0.70732366+0i
[6,] -0.04856366-0.03102418i -0.04856366+0.03102418i  0.24832494+0i
```

Note that there are four complex eigenvalues, in two conjugate pairs. The eigenvectors for the complex eigenvalues are also complex, in conjugate pairs.

Theorem 5.16. Let $\mathbf{A}$ be an $n \times n$ matrix with eigenvalues $\lambda_1, ..., \lambda_n$. Then $|\mathbf{A}| = \lambda_1 \cdots \lambda_n$.

Proof:
$|\mathbf{A}| = |\mathbf{A} - 0\mathbf{I}| = q(0) = (-1)^n(0 - \lambda_1) \cdots (0 - \lambda_n) = (-1)^{2n}\lambda_1 \cdots \lambda_n = \lambda_1 \cdots \lambda_n$.

5.7 Symmetric Matrices

Theorem 5.17. Let $\mathbf{A}$ be a symmetric matrix. Then all the eigenvalues of $\mathbf{A}$ are real.

Proof:
Let $\lambda = a + ib$ be an eigenvalue of $\mathbf{A}$, and let $\mathbf{e} = \mathbf{x} + i\mathbf{y}$ be a non-zero eigenvector for λ. We will show that $b = 0$. Since $\mathbf{e}$ is an eigenvector for λ,

$$\begin{aligned} \mathbf{Ae} &= \lambda\mathbf{e} \\ \mathbf{A}(\mathbf{x} + i\mathbf{y}) &= (a + ib)(\mathbf{x} + i\mathbf{y}) \\ \mathbf{Ax} + i\mathbf{Ay} &= a\mathbf{x} - b\mathbf{y} + i(b\mathbf{x} + a\mathbf{y}) \end{aligned}$$

Equating real and imaginary parts, we have

$$\mathbf{Ax} = a\mathbf{x} - b\mathbf{y} \qquad \mathbf{Ay} = b\mathbf{x} + a\mathbf{y}$$

Multiplying the first equation by $\mathbf{y}^T$ and the second by $\mathbf{x}^T$ yields

$$\mathbf{y}^T\mathbf{Ax} = a\mathbf{y}^T\mathbf{x} - b\mathbf{y}^T\mathbf{y}$$
$$\mathbf{x}^T\mathbf{Ay} = b\mathbf{x}^T\mathbf{x} + a\mathbf{x}^T\mathbf{y}$$

Now $\mathbf{y}^T\mathbf{Ax}$ is 1×1, and is thus equal to its transpose. We therefore have $\mathbf{y}^T\mathbf{Ax} = (\mathbf{y}^T\mathbf{Ax})^T = \mathbf{x}^T\mathbf{A}^T\mathbf{y}$. Since $\mathbf{A}$ is symmetric, $\mathbf{A}^T = \mathbf{A}$. Therefore $\mathbf{y}^T\mathbf{Ax} = \mathbf{x}^T\mathbf{Ay}$. Now, subtracting the two equations above yields

$$\begin{aligned}\mathbf{y}^T\mathbf{Ax} - \mathbf{x}^T\mathbf{Ay} &= a\mathbf{y}^T\mathbf{x} - b\mathbf{y}^T\mathbf{y} - b\mathbf{x}^T\mathbf{x} - a\mathbf{x}^T\mathbf{y} \\ 0 &= -b\mathbf{y}^T\mathbf{y} - b\mathbf{x}^T\mathbf{x} \\ 0 &= -b(\mathbf{y}^T\mathbf{y} + \mathbf{x}^T\mathbf{x}) \\ 0 &= -b|\mathbf{e}|^2\end{aligned}$$

Since $\mathbf{e} \neq \mathbf{0}$, $|\mathbf{e}|^2 > 0$. Therefore $b = 0$, so $\lambda = a$ is real.

Note that since $b = 0$, $\mathbf{Ax} = a\mathbf{x}$ and $\mathbf{Ay} = a\mathbf{y}$. It follows that if $\mathbf{e} = \mathbf{x} + i\mathbf{y}$ is a complex eigenvector for λ, then both its real and imaginary parts are real eigenvectors for λ. Therefore for a symmetric matrix, any complex eigenvector is a linear combination of two real eigenvectors (with one of the coefficients of the linear combination being i). For this reason it suffices for symmetric matrices to restrict consideration to real eigenvectors.

Theorem 5.18. Let $\mathbf{A}$ be an $n \times n$ symmetric matrix. Let $\mathbf{e}_1$ and $\mathbf{e}_2$ be eigenvectors with eigenvalues $\lambda_1 \neq \lambda_2$. Then $\mathbf{e}_1 \perp \mathbf{e}_2$.

Proof:
We show that $\mathbf{e}_1^T\mathbf{e}_2 = 0$.
$\lambda_1\mathbf{e}_1^T\mathbf{e}_2 = (\lambda_1\mathbf{e}_1)^T\mathbf{e}_2 = (\mathbf{Ae}_1)^T\mathbf{e}_2 = \mathbf{e}_1^T\mathbf{A}^T\mathbf{e}_2 = \mathbf{e}_1^T\mathbf{Ae}_2 = \mathbf{e}_1^T\lambda_2\mathbf{e}_2 = \lambda_2\mathbf{e}_1^T\mathbf{e}_2$.
Since $\lambda_1 \neq \lambda_2$, $\mathbf{e}_1^T\mathbf{e}_2 = 0$.

Theorem 5.19. Let $\mathbf{A}$ be a symmetric $n \times n$ matrix. Let λ be an eigenvalue of multiplicity k. Then there exist k orthonormal eigenvectors for λ.

Proof: Omitted.

The following theorem shows that **a symmetric matrix is determined by its eigenvectors and eigenvalues**.

Theorem 5.20. Let $\mathbf{A}$ be an $n \times n$ symmetric matrix with eigenvalues $\lambda_1, ..., \lambda_n$, and corresponding orthonormal eigenvectors $\mathbf{e}_1, ..., \mathbf{e}_n$. Let $\mathbf{\Lambda}$ be the $n \times n$ diagonal matrix whose ith diagonal element is λ_i and let $\mathbf{\Gamma}$ be the $n \times n$ orthonormal matrix whose ith column is $\mathbf{e}_i$. Then $\mathbf{A} = \mathbf{\Gamma\Lambda\Gamma}^T$.

Proof:
We first show that $\mathbf{\Gamma}^T\mathbf{A\Gamma} = \mathbf{\Lambda}$. The ith row of $\mathbf{\Gamma}^T$ is $\mathbf{e}_i^T$ and the jth column of $\mathbf{\Gamma}$ is $\mathbf{e}_j$. Therefore the ij element of $\mathbf{\Gamma}^T\mathbf{A\Gamma}$ is $\mathbf{e}_i^T\mathbf{A}\mathbf{e}_j = (\mathbf{A}^T\mathbf{e}_i)^T\mathbf{e}_j = (\mathbf{A}\mathbf{e}_i)^T\mathbf{e}_j = \lambda_i\mathbf{e}_i^T\mathbf{e}_j$. Now if $i \neq j$ then $\mathbf{e}_i^T\mathbf{e}_j = 0$ because $\mathbf{e}_i \perp \mathbf{e}_j$. Therefore the off-diagonal elements of $\mathbf{\Gamma}^T\mathbf{A\Gamma}$ are 0. The ith diagonal element of $\mathbf{\Gamma}^T\mathbf{A\Gamma}$ is $\lambda_i\mathbf{e}_i^T\mathbf{e}_i = \lambda_i$, because $\mathbf{e}_i^T\mathbf{e}_i = 1$. Therefore $\mathbf{\Gamma}^T\mathbf{A\Gamma} = \mathbf{\Lambda}$.

Now since $\mathbf{\Gamma}$ is orthonormal, $\mathbf{\Gamma}^T\mathbf{\Gamma} = \mathbf{\Gamma\Gamma}^T = \mathbf{I}$. Therefore $\mathbf{\Gamma\Lambda\Gamma}^T = \mathbf{\Gamma}(\mathbf{\Gamma}^T\mathbf{A\Gamma})\mathbf{\Gamma}^T = \mathbf{IAI} = \mathbf{A}$.

Definition 5.29. The decomposition $\mathbf{A} = \mathbf{\Gamma\Lambda\Gamma}^T$ is called the **eigendecomposition** of $\mathbf{A}$.

Example 5.16. Find the eigendecomposition of $\begin{bmatrix} 4.6 & 7.2 \\ 7.2 & 0.4 \end{bmatrix}$.

Solution:
First we must find the eigenvalues and eigenvectors. The eigenvalues are the roots of

$$\begin{vmatrix} 4.6 - \lambda & 7.2 \\ 7.2 & 0.4 - \lambda \end{vmatrix}$$

We solve $(4.6 - \lambda)(0.4 - \lambda) - 7.2^2 = 0$. Multiplying out, we get $\lambda^2 - 5\lambda - 50 = 0$. The roots are $\lambda_1 = 10$, $\lambda_2 = -5$.

Now we find the eigenvectors. First we solve $\begin{bmatrix} 4.6 & 7.2 \\ 7.2 & 0.4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 10 \begin{bmatrix} x \\ y \end{bmatrix}$.

Any vector of the form $(4x, 3x)$ is a solution. A standardized eigenvector is $(0.8, 0.6)^T$ [we could also use $(-0.8, -0.6)^T$].

Now we solve $\begin{bmatrix} 4.6 & 7.2 \\ 7.2 & 0.4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = -5 \begin{bmatrix} x \\ y \end{bmatrix}$.

Any vector of the form $(-3x, 4x)$ is a solution. A standardized eigenvector is $(-0.6, 0.8)$ [we could also use $(0.6, -0.8)$]. The eigendecomposition is

$$\begin{bmatrix} 4.6 & 7.2 \\ 7.2 & 0.4 \end{bmatrix} = \begin{bmatrix} 0.8 & -0.6 \\ 0.6 & 0.8 \end{bmatrix} \begin{bmatrix} 10 & 0 \\ 0 & -5 \end{bmatrix} \begin{bmatrix} 0.8 & 0.6 \\ -0.6 & 0.8 \end{bmatrix}$$

We can write the eigendecomposition in another way. For the matrix in Example 5.16, note that

$$\begin{bmatrix} 4.6 & 7.2 \\ 7.2 & 0.4 \end{bmatrix} = 10 \begin{bmatrix} 0.8 \\ 0.6 \end{bmatrix} \begin{bmatrix} 0.8 & 0.6 \end{bmatrix} - 5 \begin{bmatrix} 0.8 \\ -0.6 \end{bmatrix} \begin{bmatrix} 0.8 & -0.6 \end{bmatrix}$$

The following corollary states that this holds in general.

Corollary:
Let $\mathbf{A}$ be an $n \times n$ symmetric matrix with eigenvalues $\lambda_1, ..., \lambda_n$, and corresponding orthonormal eigenvectors $\mathbf{e}_1, ..., \mathbf{e}_n$. Then

$$\mathbf{A} = \sum_{i=1}^{n} \lambda_i \mathbf{e}_i \mathbf{e}_i^T$$

Theorem 5.21. Let $\mathbf{A}$ be a symmetric matrix with eigenvalues $\lambda_1, ..., \lambda_n$.
Then $tr(\mathbf{A}) = \lambda_1 + \cdots + \lambda_n$.

Proof:
Let $\mathbf{A} = \mathbf{\Gamma \Lambda \Gamma}^T$ be the eigendecomposition of $\mathbf{A}$. Since $\mathbf{\Gamma}$ is orthonormal, $\mathbf{\Gamma}^T\mathbf{\Gamma} = \mathbf{I}$.
Now $tr(\mathbf{A}) = tr(\mathbf{\Gamma \Lambda \Gamma}^T) = tr(\mathbf{\Gamma}^T\mathbf{\Gamma \Lambda}) = tr(\mathbf{I \Lambda}) = tr(\mathbf{\Lambda}) = \lambda_1 + \cdots + \lambda_n$.

The eigendecomposition makes it easy to find powers of symmetric matrices.

Theorem 5.22. Let $\mathbf{A}$ be a symmetric matrix with eigendecomposition $\mathbf{A} = \mathbf{\Gamma \Lambda \Gamma}^T$. Let n be any positive integer. Then $\mathbf{A}^n = \mathbf{\Gamma \Lambda}^n \mathbf{\Gamma}^T$.

Proof: Since $\mathbf{\Gamma}$ is orthonormal, $\mathbf{\Gamma}^T\mathbf{\Gamma} = \mathbf{I}$.
Therefore $\mathbf{A}^2 = \mathbf{\Gamma \Lambda \Gamma}^T\mathbf{\Gamma \Lambda \Gamma}^T = \mathbf{\Gamma \Lambda I \Lambda \Gamma}^T = \mathbf{\Gamma \Lambda \Lambda \Gamma}^T = \mathbf{\Gamma \Lambda}^2\mathbf{\Gamma}^T$. The result follows by induction.

Note that Λ^n is the diagonal matrix with ith diagonal element λ_i^n.

The eigendecomposition makes it easy to find inverses of symmetric matrices, when they exist.

Theorem 5.23. Let $\mathbf{A}$ be a symmetric matrix with eigendecomposition $\mathbf{A} = \mathbf{\Gamma\Lambda\Gamma}^T$, where all the eigenvalues are positive. Then $\mathbf{A}$ is nonsingular, and $\mathbf{A}^{-1} = \mathbf{\Gamma\Lambda}^{-1}\mathbf{\Gamma}^T$.

Proof: Since $\mathbf{\Gamma}$ is orthonormal, $\mathbf{\Gamma}^T\mathbf{\Gamma\Gamma}^T = \mathbf{I}$.
Now $\mathbf{A}(\mathbf{\Gamma\Lambda}^{-1}\mathbf{\Gamma}^T) = \mathbf{\Gamma\Lambda\Gamma}^T\mathbf{\Gamma\Lambda}^{-1}\mathbf{\Gamma}^T = \mathbf{\Gamma\Lambda I\Lambda}^{-1}\mathbf{\Gamma}^T = \mathbf{\Gamma\Lambda\Lambda}^{-1}\mathbf{\Gamma}^T = \mathbf{\Gamma I\Gamma}^T = \mathbf{\Gamma\Gamma}^T = \mathbf{I}$.

Definition 5.30. Let $\mathbf{A}$ be a symmetric matrix with eigendecomposition $\mathbf{A} = \mathbf{\Gamma\Lambda\Gamma}^T$. If all the eigenvalues are nonnegative, then for any positive real number n we define $\mathbf{A}^n = \mathbf{\Gamma\Lambda}^n\mathbf{\Gamma}^T$.

For example, the square root of $\mathbf{A}$ is $\mathbf{A}^{1/2} = \mathbf{\Gamma\Lambda}^{1/2}\mathbf{\Gamma}^T$. The square root is real if and only if the diagonal elements of $\mathbf{\Lambda}$ (i.e., the eigenvalues) are nonnegative. Note that $\mathbf{A}^{1/2}\mathbf{A}^{1/2} = \mathbf{A}$.

5.8 Quadratic Forms

Definition 5.31. Let $\mathbf{A}$ be an $n \times n$ symmetric matrix. A **quadratic form** in $\mathbf{A}$ is the function of n variables $\mathbf{x} = (x_1, ..., x_n)^T$ defined by

$$Q(\mathbf{x}) = \mathbf{x}^T\mathbf{A}\mathbf{x}$$

Multiplying out yields the equivalent formulation

$$Q(\mathbf{x}) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_ix_j$$

Note that $Q(\mathbf{0}) = 0$.

Definition 5.32. The symmetric matrix $\mathbf{A}$ is **positive definite** if $\mathbf{x}^T\mathbf{A}\mathbf{x} > 0$ for all $\mathbf{x} \neq 0$. $\mathbf{A}$ is **nonnegative definite** if $\mathbf{x}^T\mathbf{A}\mathbf{x} \geq 0$ for all $\mathbf{x}$.

Theorem 5.24. Let $\mathbf{A}$ be an $n \times n$ symmetric matrix. Then
1) $\mathbf{A}$ is positive definite if and only if all its eigenvalues are positive.
2) $\mathbf{A}$ is nonnegative definite if and only if all its eigenvalues are nonnegative.

Proof:
First assume that there exists a negative eigenvalue λ_i. Let $\mathbf{e}_i$ be the standardized eigenvector associated with λ_i. Then $\mathbf{e}_i^T\mathbf{A}\mathbf{e}_i = \mathbf{e}_i^T\lambda_i\mathbf{e}_i = \lambda_i\mathbf{e}_i^T\mathbf{e}_i = \lambda_i < 0$, so $\mathbf{A}$ is neither positive definite nor nonnegative definite.

Now assume that all the eigenvalues are nonnegative. Let $\mathbf{x} \neq 0$ be an $n \times 1$ vector. Let $\mathbf{A} = \mathbf{\Gamma}\mathbf{\Lambda}\mathbf{\Gamma}^T$ be the eigendecomposition of $\mathbf{A}$. Let $\mathbf{y} = \mathbf{\Gamma}^T\mathbf{x}$. Since $\mathbf{\Gamma}$ is an orthogonal matrix, it is of full rank, so $\mathbf{y} \neq \mathbf{0}$. Then
$\mathbf{x}^T\mathbf{A}\mathbf{x} = \mathbf{x}^T\mathbf{\Gamma}\mathbf{\Lambda}\mathbf{\Gamma}^T\mathbf{x} = (\mathbf{\Gamma}^T\mathbf{x})^T\mathbf{\Lambda}(\mathbf{\Gamma}^T\mathbf{x}) = \mathbf{y}^T\mathbf{\Lambda}\mathbf{y} = \sum_{i=1}^n \lambda_i \mathbf{y}_i^2$.

If $\lambda_i > 0$ for all i, $\sum_{i=1}^n \lambda_i \mathbf{y}_i^2 > 0$, which shows that $\mathbf{A}$ is positive definite. If $\lambda_i \geq 0$ for all i, $\sum_{i=1}^n \lambda_i \mathbf{y}_i^2 \geq 0$, which shows that $\mathbf{A}$ is nonnegative definite.

Theorem 5.25. Let $\mathbf{X}$ be a matrix whose columns are linearly independent. Then all the eigenvalues of the matrix $\mathbf{X}^T\mathbf{X}$ are positive.

Proof:
We show that $\mathbf{X}^T\mathbf{X}$ is positive definite. Let $\mathbf{v} \neq \mathbf{0}$ be any non-zero vector. Let $\mathbf{a} = \mathbf{X}\mathbf{v}$. Since the columns of $\mathbf{X}$ are linearly independent, $\mathbf{a} \neq \mathbf{0}$. Then $\mathbf{a}^T\mathbf{X}^T\mathbf{X}\mathbf{a} = \mathbf{v}^T\mathbf{v} = |\mathbf{v}|^2 > 0$. It follows from Theorem 5.24 that all the eigenvalues of $\mathbf{X}^T\mathbf{X}$ are positive.

5.9 Singular Value Decomposition

The singular value decomposition extends the idea of a eigendecomposition to non-square matrices. We will develop the singular value decomposition for full rank matrices whose number of rows is greater than or equal to the number of columns. A similar development can be made for matrices with a greater number of columns than rows.

Theorem 5.26. Let $\mathbf{X}$ be an $n \times p$ full rank matrix, with $n \geq p$. Then $\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}\mathbf{X}^T$ are both symmetric. $\mathbf{X}^T\mathbf{X}$ is positive definite and $\mathbf{X}\mathbf{X}^T$ is nonnegative definite.

Proof:
It is clear that $\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}\mathbf{X}^T$ are both symmetric. Let $\mathbf{v} \neq \mathbf{0} \in \mathcal{R}^p$. Then $\mathbf{X}\mathbf{v} \neq \mathbf{0}$ because $\mathbf{X}$ is of full rank. Thus $\mathbf{v}^T\mathbf{X}^T\mathbf{X}\mathbf{v} = (\mathbf{X}\mathbf{v})^T\mathbf{X}\mathbf{v} = |\mathbf{X}\mathbf{v}|^2 > 0$. Now let $\mathbf{w} \in \mathcal{R}^n$. Then $\mathbf{w}^T\mathbf{X}\mathbf{X}^T\mathbf{w} = (\mathbf{X}^T\mathbf{w})^T\mathbf{X}^T\mathbf{w} = |\mathbf{X}^T\mathbf{w}|^2 \geq 0$.

We now derive the singular value decomposition for a $n \times p$ matrix. The derivation differs slightly depending on whether $n \geq p$ or $n < p$. We present the derivation for $n \geq p$, then describe how it may be modified for the case $n < p$.

Theorem 5.27. Let $\mathbf{X}$ be an $n \times p$ full rank matrix, with $n \geq p$. Let $\mathbf{e}_1, ..., \mathbf{e}_p$ be the standardized eigenvectors of $\mathbf{X}^T\mathbf{X}$ with eigenvalues $\lambda_1, ..., \lambda_p$. Since $\mathbf{X}^T\mathbf{X}$ is positive definite, all the λ_i are positive, so all the values $\lambda_i^{-1/2}$ are real. For each i, let $\mathbf{u}_i = \lambda_i^{-1/2}\mathbf{X}\mathbf{e}_i$. Then $\mathbf{u}_1, ..., \mathbf{u}_p$ are standardized eigenvectors of $\mathbf{X}\mathbf{X}^T$ with eigenvalues $\lambda_1, ..., \lambda_p$.

Proof:
We first show that $\mathbf{u}_i^T\mathbf{u}_i = 1$.
$\mathbf{u}_i^T\mathbf{u}_i = \mathbf{e}_i^T\mathbf{X}^T\lambda_i^{-1/2}\lambda_i^{-1/2}\mathbf{X}\mathbf{e}_i = \lambda^{-1}\mathbf{e}_i^T\mathbf{X}^T\mathbf{X}\mathbf{e}_i = \lambda^{-1}\mathbf{e}_i^T\lambda\mathbf{e}_i = \mathbf{e}_i^T\mathbf{e}_i = 1.$

Now let $\mathbf{e}_i$ be an eigenvector of $\mathbf{X}^T\mathbf{X}$ with eigenvalue λ_i. Let $\mathbf{u}_i = \lambda_i^{-1/2}\mathbf{X}\mathbf{e}_i$. Then
$(\mathbf{X}\mathbf{X}^T)\mathbf{u}_i = (\mathbf{X}\mathbf{X}^T)\lambda_i^{-1/2}\mathbf{X}\mathbf{e}_i = \mathbf{X}\lambda_i^{-1/2}(\mathbf{X}^T\mathbf{X}\mathbf{e}_i) = \mathbf{X}\lambda_i^{-1/2}\lambda_i\mathbf{e}_i = \lambda_i^{1/2}(\mathbf{X}\mathbf{e}_i) = \lambda_i\mathbf{u}_i.$

Theorem 5.28. Let $\mathbf{X}$ be an $n \times p$ matrix with $n \geq p$. Let $\mathbf{\Gamma}$ be the $p \times p$ matrix whose columns are the standardized eigenvectors of $\mathbf{X}^T\mathbf{X}$, let $\lambda_1, ..., \lambda_p$ be the eigenvalues of those eigenvectors, let $\mathbf{L}$ be the diagonal matrix whose ith diagonal element is $\sqrt{\lambda_i}$, and let $\mathbf{U}$ be the $n \times p$ matrix of eigenvectors of $\mathbf{X}\mathbf{X}^T$ as defined above. Then

$$\mathbf{X} = \mathbf{U}\mathbf{L}\mathbf{\Gamma}^T$$

so $\mathbf{X}$ can be written as the product of an orthonormal $n \times p$ matrix $\mathbf{U}$, a $p \times p$ diagonal matrix $\mathbf{L}$, and an orthonormal $p \times p$ matrix $\mathbf{\Gamma}^T$.

Proof:
Since the ith column of $\mathbf{U}$ is $\mathbf{u}_i = \lambda_i^{-1/2}\mathbf{X}\mathbf{e}_i$, it follows that $\mathbf{U} = \mathbf{X}\mathbf{\Gamma}\mathbf{L}^{-1}$. Now $\mathbf{U}$ and $\mathbf{\Gamma}$ are orthonormal because their columns are standardized eigenvectors of symmetric matrices. $\mathbf{L}$ is diagonal by construction. Note that $\mathbf{\Gamma}\mathbf{\Gamma}^T = \mathbf{I}$. It follows that
$\mathbf{U}\mathbf{L}\mathbf{\Gamma}^T = \mathbf{X}\mathbf{\Gamma}\mathbf{L}^{-1}\mathbf{L}\mathbf{\Gamma}^T = \mathbf{X}\mathbf{\Gamma}\mathbf{\Gamma}^T = \mathbf{X}$.

If $n < p$, then interchanging the definitions of $\mathbf{U}$ and $\mathbf{\Gamma}$ above we can construct $\mathbf{U}$, $\mathbf{L}$, and $\mathbf{\Gamma}$ such that $\mathbf{X}^T = \mathbf{\Gamma}\mathbf{L}\mathbf{U}^T$. Then $\mathbf{X} = \mathbf{U}\mathbf{L}\mathbf{\Gamma}^T$. Here the columns of $\mathbf{U}$ are the standardized eigenvectors of $\mathbf{X}\mathbf{X}^T$, $\mathbf{L}$ is the diagonal matrix of the square roots of the corresponding eigenvalues $\sqrt{\lambda_i}$, and $\mathbf{e}_i = \lambda_i^{-1/2}\mathbf{X}^T\mathbf{u}_i$. The $\mathbf{e}_i$ are standardized eigenvectors of $\mathbf{X}^T\mathbf{X}$, with eigenvalues $\lambda_1, ..., \lambda_p$.

Definition 5.33. The decomposition $\mathbf{X} = \mathbf{U}\mathbf{L}\mathbf{\Gamma}^T$ is called the **singular value decomposition** of $\mathbf{X}$. The diagonal elements of $\mathbf{L}$, $\sqrt{\lambda_1}, ..., \sqrt{\lambda_p}$, are called the **singular values** of $\mathbf{X}$. The columns of $\mathbf{U}$ are called the **left singular vectors** and the columns of $\mathbf{\Gamma}$ are called the **right singular vectors**.

Example 5.17. Find the singular value decomposition of the matrix $\mathbf{X} = \begin{bmatrix} 3 & -1 \\ 1 & 3 \\ 1 & 1 \end{bmatrix}$.

Solution:

We first compute $\mathbf{X}^T\mathbf{X}$: $\mathbf{X}^T\mathbf{X} = \begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix} \begin{bmatrix} 3 & -1 \\ 1 & 3 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 11 & 1 \\ 1 & 11 \end{bmatrix}$.

The eigenvalues of $\mathbf{X}^T\mathbf{X}$ solve the characteristic equation $\lambda^2 - 22\lambda + 120 = (\lambda - 12)(\lambda - 10) = 0$, so the eigenvalues are $\lambda_1 = 12$, $\lambda_2 = 10$.

Solving $\begin{bmatrix} 11 & 1 \\ 1 & 11 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 12 \begin{bmatrix} x \\ y \end{bmatrix}$ and $\begin{bmatrix} 11 & 1 \\ 1 & 11 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 10 \begin{bmatrix} x \\ y \end{bmatrix}$

yields the standardized eigenvectors $\mathbf{e}_1 = (1/\sqrt{2}, 1/\sqrt{2})^T$ and $\mathbf{e}_2 = (1/\sqrt{2}, -1/\sqrt{2})^T$.

We now have $\mathbf{\Gamma} = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}$ and $\mathbf{L} = \begin{bmatrix} \sqrt{12} & 0 \\ 0 & \sqrt{10} \end{bmatrix}$.

Therefore $\mathbf{U} = \mathbf{X}\mathbf{\Gamma}\mathbf{L}^{-1} = \begin{bmatrix} 1/\sqrt{6} & 2/\sqrt{5} \\ 2/\sqrt{6} & -1/\sqrt{5} \\ 1/\sqrt{6} & 0 \end{bmatrix}$.

The singular value decomposition is

$$\begin{bmatrix} 3 & -1 \\ 1 & 3 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{6} & 2/\sqrt{5} \\ 2/\sqrt{6} & -1/\sqrt{5} \\ 1/\sqrt{6} & 0 \end{bmatrix} \begin{bmatrix} \sqrt{12} & 0 \\ 0 & \sqrt{10} \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}$$

5.10 Another Look at Linear Regression

The linear model is

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i \quad \text{for } i = 1, ..., n$$

where $\varepsilon_1, ..., \varepsilon_n$ are independent random variables with mean 0 and variance σ^2.

The linear regression estimate $\hat{\beta}$ minimizes

$$S = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2$$

The partial derivatives with respect to β_k are therefore equal to 0 for $k = 0, ..., p$. We compute the partial derivative with respect to β_0.

$$\frac{\partial S}{\partial \beta_0} = \sum_{i=1}^{n} -2 \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right) = 0$$

It follows that

$$\hat{\beta}_0 = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{p} x_{ij} \hat{\beta}_j \right) = \bar{y} - \sum_{j=1}^{p} \bar{x}_{.j} \hat{\beta}_j$$

Once we compute estimates $\hat{\beta}_1, ..., \hat{\beta}_p$, we can substitute these values to compute $\hat{\beta}_0$. We now describe how to compute $\hat{\beta}_1, ..., \hat{\beta}_p$.

Substituting the expression for β_0 into the expression for S yields

$$S = \sum_{i=1}^{n} \left(y_i - \bar{y} - \sum_{j=1}^{p} \beta_j (x_{ij} - \bar{x}_{.j}) \right)^2$$

This has the form of the sum of squared residuals from a model in which the vector of observed variables is $\mathbf{Y} - \bar{\mathbf{Y}}$, where $\bar{\mathbf{Y}}$ is the $n \times 1$ vector all of whose elements are $\bar{y}$, and the matrix $\mathbf{X}$ of predictor variables is the $n \times p$ matrix whose ij element is $x_{ij} - \bar{x}_{.j}$. Note that $\mathbf{X}$ is the usual design matrix with the intercept removed and the remaining columns centered so that they have mean 0. Let $\boldsymbol{\beta} = (\beta_1, ..., \beta_p)^T$. Note that β_0 has been removed. It follows that the values $\mathbf{X}\hat{\boldsymbol{\beta}}$ are given by

$$\begin{aligned} \mathbf{X}\hat{\boldsymbol{\beta}} &= \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T(\mathbf{Y} - \bar{\mathbf{Y}}) \\ &= \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} - \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\bar{\mathbf{Y}} \end{aligned}$$

Now since each of the columns of $\mathbf{X}$ sum to 0, $\mathbf{X}^T\bar{\mathbf{Y}} = \mathbf{0}$. The values $\mathbf{X}\hat{\boldsymbol{\beta}}$ are thus given by

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

We rewrite this, using the singular value decomposition $\mathbf{X} = \mathbf{U}\mathbf{L}\boldsymbol{\Gamma}^T$. Note that $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ and $\boldsymbol{\Gamma}^T\boldsymbol{\Gamma} = \mathbf{I}$. Then

$$\begin{aligned} \mathbf{X}\hat{\boldsymbol{\beta}} &= \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \\ &= \mathbf{U}\mathbf{L}\boldsymbol{\Gamma}^T(\boldsymbol{\Gamma}\mathbf{L}\mathbf{U}^T\mathbf{U}\mathbf{L}\boldsymbol{\Gamma}^T)^{-1}\boldsymbol{\Gamma}\mathbf{L}\mathbf{U}^T\mathbf{Y} \end{aligned}$$

$$
\begin{aligned}
&= \mathbf{U}\mathbf{L}\boldsymbol{\Gamma}^T(\boldsymbol{\Gamma}\mathbf{L}\mathbf{L}\boldsymbol{\Gamma}^T)^{-1}\boldsymbol{\Gamma}\mathbf{L}\mathbf{U}^T\mathbf{Y} \\
&= \mathbf{U}\mathbf{L}\boldsymbol{\Gamma}^T(\boldsymbol{\Gamma}\mathbf{L}^2\boldsymbol{\Gamma}^T)^{-1}\boldsymbol{\Gamma}\mathbf{L}\mathbf{U}^T\mathbf{Y} \\
&= \mathbf{U}\mathbf{L}(\boldsymbol{\Gamma}^T\boldsymbol{\Gamma})\mathbf{L}^{-2}(\boldsymbol{\Gamma}^T\boldsymbol{\Gamma})\mathbf{L}\mathbf{U}^T\mathbf{Y} \\
&= \mathbf{U}\mathbf{L}\mathbf{L}^{-2}\mathbf{L}\mathbf{U}^T\mathbf{Y} \\
&= \mathbf{U}\mathbf{U}^T\mathbf{Y}
\end{aligned}
$$

Note that $\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T = \mathbf{U}\mathbf{U}^T$.

Let $\mathbf{u}_i$ be the ith column of $\mathbf{U}$. Then the linear regression estimate can be written

$$\mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{U}\mathbf{U}^T\mathbf{Y} = \sum_{i=1}^{p} \mathbf{u}_i(\mathbf{u}_i^T\mathbf{Y})$$

Thus $\mathbf{X}\hat{\boldsymbol{\beta}}$ is a linear combination of the columns of $\mathbf{U}$, with the coefficient of the ith column being $\mathbf{u}_i^T\mathbf{Y}$.

Now we'll calculate the error in estimating $\mathbf{X}\boldsymbol{\beta}$.

$$
\begin{aligned}
\mathbf{X}\hat{\boldsymbol{\beta}} &= \mathbf{U}\mathbf{U}^T\mathbf{Y} \\
&= \mathbf{U}\mathbf{U}^T(\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}) \\
&= \mathbf{U}\mathbf{U}^T(\mathbf{U}\mathbf{L}\boldsymbol{\Gamma}^T\boldsymbol{\beta} + \boldsymbol{\varepsilon}) \\
&= \mathbf{U}(\mathbf{U}^T\mathbf{U})\mathbf{L}\boldsymbol{\Gamma}^T\boldsymbol{\beta} + \mathbf{U}\mathbf{U}^T\boldsymbol{\varepsilon} \\
&= \mathbf{U}\mathbf{L}\boldsymbol{\Gamma}^T\boldsymbol{\beta} + \mathbf{U}\mathbf{U}^T\boldsymbol{\varepsilon} \\
&= \mathbf{X}\boldsymbol{\beta} + \mathbf{U}\mathbf{U}^T\boldsymbol{\varepsilon}
\end{aligned}
$$

It follows that $E(\mathbf{X}\hat{\boldsymbol{\beta}}) = E(\mathbf{X}\boldsymbol{\beta} + \mathbf{U}\mathbf{U}^T\boldsymbol{\varepsilon}) = \mathbf{X}\boldsymbol{\beta} + \mathbf{U}\mathbf{U}^T E(\boldsymbol{\varepsilon}) = \mathbf{X}\boldsymbol{\beta}$, since $E(\boldsymbol{\varepsilon}) = \mathbf{0}$. The error in estimating $\mathbf{X}\boldsymbol{\beta}$ is $\mathbf{X}\hat{\boldsymbol{\beta}} - \mathbf{X}\boldsymbol{\beta} = \mathbf{U}\mathbf{U}^T\boldsymbol{\varepsilon}$. We now compute the mean squared error. Let $\mathbf{H} = \mathbf{U}\mathbf{U}^T$.

$$
\begin{aligned}
\text{MSE} &= E(||\mathbf{U}\mathbf{U}^T\boldsymbol{\varepsilon}||^2) \\
&= E(\boldsymbol{\varepsilon}^T\mathbf{U}\mathbf{U}^T\mathbf{U}\mathbf{U}^T\boldsymbol{\varepsilon}) \\
&= E(\boldsymbol{\varepsilon}^T\mathbf{U}\mathbf{U}^T\boldsymbol{\varepsilon}) \\
&= E(\boldsymbol{\varepsilon}^T\mathbf{H}\boldsymbol{\varepsilon}) \\
&= E\left(\sum_{i=1}^{n}\sum_{j=1}^{n} h_{ij}\varepsilon_i\varepsilon_j\right)
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{i=1}^{n}\sum_{j=1}^{n} h_{ij}E(\varepsilon_i\varepsilon_j) \\
&= \sum_{i=1}^{n} h_{ii}E(\varepsilon_i^2) + \sum_{i=1}^{n}\sum_{j\neq i} h_{ij}E(\varepsilon_i\varepsilon_j) \\
&= \sum_{i=1}^{n} h_{ii}\sigma^2 + \sum_{i=1}^{n}\sum_{j\neq i}(0) \\
&= tr(\mathbf{H})\sigma^2 \\
&= tr(\mathbf{U}\mathbf{U}^T)\sigma^2 \\
&= tr(\mathbf{U}^T\mathbf{U})\sigma^2 \\
&= tr(\mathbf{I}_{p\times p})\sigma^2 \\
&= p\sigma^2
\end{aligned}
$$

5.11 Another Look at Ridge Regression

We can use the singular value decomposition to get insight into the difference between ridge regression and linear regression. We know that the ridge regression estimate $\hat{\boldsymbol{\beta}} = (\hat{\beta}_1, ..., \hat{\beta}_p)^T$, is given by

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}$$

The estimate of $\mathbf{X}\boldsymbol{\beta}$ is

$$\mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}$$

Now the singular value decomposition for $\mathbf{X}$ is $\mathbf{X} = \mathbf{U}\mathbf{L}\boldsymbol{\Gamma}^T$, where $\mathbf{L}$ is a diagonal matrix whose diagonal elements are the square roots of the eigenvalues of the columns of $\mathbf{U}$ (and of $\boldsymbol{\Gamma}$). Recall that $\boldsymbol{\Gamma}^T\boldsymbol{\Gamma} = \mathbf{I}$ and $\mathbf{U}^T\mathbf{U} = \mathbf{I}$. We denote these eigenvalues $d_1^2, ..., d_p^2$. We usually call them $\lambda_1, ..., \lambda_p$, but this might cause confusion with the ridge parameter λ. So the diagonal elements of $\mathbf{L}$ are $d_1, ..., d_p$.

Substituting $\mathbf{X} = \mathbf{U}\mathbf{L}\boldsymbol{\Gamma}^T$ into the expression for $\mathbf{X}\hat{\boldsymbol{\beta}}$ yields

$$
\begin{aligned}
\mathbf{X}\hat{\boldsymbol{\beta}} &= \mathbf{U}\mathbf{L}\boldsymbol{\Gamma}^T(\boldsymbol{\Gamma}\mathbf{L}\mathbf{U}^T\mathbf{U}\mathbf{L}\boldsymbol{\Gamma}^T + \lambda\mathbf{I})^{-1}\boldsymbol{\Gamma}\mathbf{L}\mathbf{U}^T\mathbf{Y} \\
&= \mathbf{U}\mathbf{L}\boldsymbol{\Gamma}^T(\boldsymbol{\Gamma}\mathbf{L}^2\boldsymbol{\Gamma}^T + \lambda\mathbf{I})^{-1}\boldsymbol{\Gamma}\mathbf{L}\mathbf{U}^T\mathbf{Y} \\
&= \mathbf{U}\mathbf{L}\boldsymbol{\Gamma}^T(\boldsymbol{\Gamma}\mathbf{L}^2\boldsymbol{\Gamma}^T + \lambda\boldsymbol{\Gamma}^T\boldsymbol{\Gamma}\mathbf{I})^{-1}\boldsymbol{\Gamma}\mathbf{L}\mathbf{U}^T\mathbf{Y} \\
&= \mathbf{U}\mathbf{L}\boldsymbol{\Gamma}^T(\boldsymbol{\Gamma}\mathbf{L}^2\boldsymbol{\Gamma}^T + \boldsymbol{\Gamma}^T\lambda\mathbf{I}\boldsymbol{\Gamma})^{-1}\boldsymbol{\Gamma}\mathbf{L}\mathbf{U}^T\mathbf{Y} \\
&= \mathbf{U}\mathbf{L}\boldsymbol{\Gamma}^T(\boldsymbol{\Gamma}(\mathbf{L}^2 + \lambda\mathbf{I})\boldsymbol{\Gamma}^T)^{-1}\boldsymbol{\Gamma}\mathbf{L}\mathbf{U}^T\mathbf{Y}
\end{aligned}
$$

$$
\begin{aligned}
&= \mathbf{U}\mathbf{L}\mathbf{\Gamma}^T(\mathbf{\Gamma}^T)^{-1}(\mathbf{L}^2+\lambda\mathbf{I})^{-1}\mathbf{\Gamma}^{-1}\mathbf{\Gamma}\mathbf{L}\mathbf{U}^T\mathbf{Y} \\
&= \mathbf{U}\mathbf{L}(\mathbf{L}^2+\lambda\mathbf{I})^{-1}\mathbf{L}\mathbf{U}^T\mathbf{Y}
\end{aligned}
$$

Now the matrix $\mathbf{L}(\mathbf{L}^2+\lambda\mathbf{I})^{-1}\mathbf{L}$ is a $p \times p$ diagonal matrix whose ith diagonal element is $\frac{d_i^2}{d_i^2+\lambda}$. Let $\mathbf{D}_\lambda = \mathbf{L}(\mathbf{L}^2+\lambda\mathbf{I})^{-1}\mathbf{L}$. Then

$$
\mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{U}\mathbf{D}_\lambda\mathbf{U}^T\mathbf{Y}
$$

The ridge regression estimate can be written

$$
\mathbf{X}\hat{\boldsymbol{\beta}}_{RR} = \mathbf{U}\mathbf{D}_\lambda\mathbf{U}^T\mathbf{Y} = \sum_{i=1}^{p}\mathbf{u}_i\left(\frac{d_i^2}{d_i^2+\lambda}\mathbf{u}_i^T\mathbf{Y}\right)
$$

We now compare this with the linear regression estimate. Recall that the linear regression estimate of $\mathbf{X}\boldsymbol{\beta}$ can be written

$$
\mathbf{X}\hat{\boldsymbol{\beta}}_{LR} = \mathbf{U}\mathbf{U}^T\mathbf{Y} = \sum_{i=1}^{p}\mathbf{u}_i(\mathbf{u}_i^T\mathbf{Y})
$$

which is a linear combination of the columns of $\mathbf{U}$, with the coefficient of the ith column being $\mathbf{u}_i^T\mathbf{Y}$.

Thus $\mathbf{X}\hat{\boldsymbol{\beta}}_{RR}$ is also a linear combination of the columns of $\mathbf{U}$, but with the coefficient of the ith column being $\left(\frac{d_i^2}{d_i^2+\lambda}\mathbf{u}_i^T\mathbf{Y}\right)$.

We see that the ridge regression estimate is obtained from the linear regression estimate by shrinking the coefficient of $\mathbf{u}_i$ by the factor $\frac{d_i^2}{d_i^2+\lambda}$. Now $\frac{d_i^2}{d_i^2+\lambda}$ is a decreasing function of d_i, so columns $\mathbf{u}_i$ whose eigenvalues are smaller are shrunk more. Thus the ridge regression estimate shrinks the linear regression estimate, with the shrinkage being greater in directions with smaller eigenvalues.

6 Principal Components

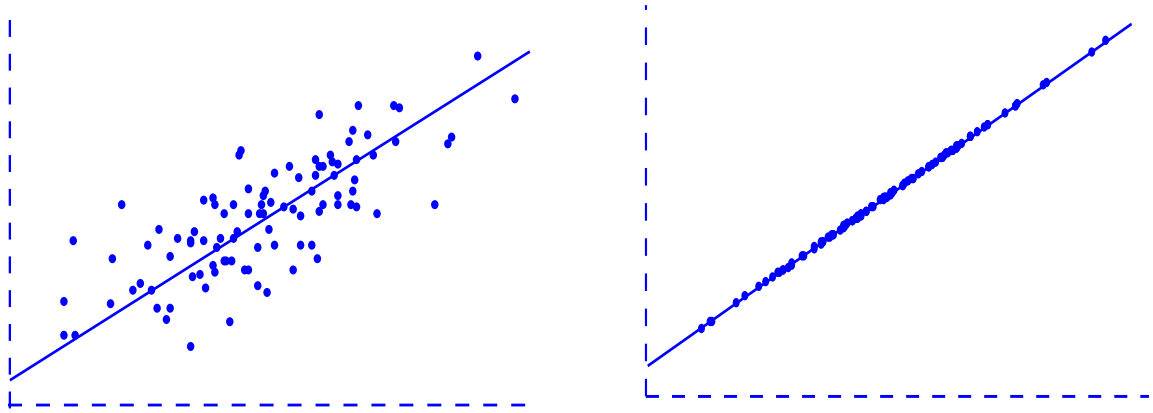
6.1 Introduction

Principal component analysis is a method by which most of the variation in a set of variables can be retained in a smaller number of linear combinations of those variables.

The following figure illustrates the idea. Most of the variation in the data is due to variation in the x-values. If we replace each two-dimensional point (x, y) with the one-dimensional point x, we retain most of the variation with a smaller number of variables.

Left: A data set containing bivariate data (x, y). *Right:* The x-values from the data set. Because most of the variation in the data is due to variation in x, we can retain most of the variation while reducing the number of variables.

Here is another example, in which the data are clustered around a line not parallel to an axis. In this case, we project the points onto the direction of greatest variation. The two-dimensional points are now one-dimensional, but most of the variation is retained.



In general we will have several variables, and we will not need to reduce the dimension all the way to 1. To develop a general method we must learn how to determine the directions of greatest variation.

6.2 Principal Components

Definition 6.1. Let $\mathbf{X}$ be a matrix each of whose columns sum to 0 (equivalently, they have mean 0). We refer to such a matrix as a **centered matrix**.

Recall that a matrix whose columns are linearly independent is said to have **full rank**.

Note:
Let $\mathbf{x} = (x_1, ..., x_n)^T$ be a column of a centered matrix $\mathbf{X}$. The squared length of $\mathbf{x}$ is $\sum_{i=1}^n x_i^2$. Because $\bar{x} = 0$, the sample variance of the elements of $\mathbf{x}$ is $\sum_{i=1}^n x_i^2/(n-1)$. So the squared length is proportional to the variance. The important idea is that a the length of a column reflects how spread out the elements are. Since any linear combination of columns of a centered matrix is centered, this holds for vectors of the form $\mathbf{Xa}$ as well.

Let $\mathbf{X}$ be a $n \times p$ centered matrix of full rank. We will find the vector $\mathbf{a}$ for which $\mathbf{Xa}$ has the greatest length, subject to the constraint $|\mathbf{a}| = 1$. Note that $\mathbf{Xa}$ is a linear combination of the columns of $\mathbf{X}$. The following theorem gives the result.

Theorem 6.1. Let $\mathbf{X}$ be an $n \times p$ centered matrix of full rank. Let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p > 0$ be the eigenvalues of $\mathbf{X}^T\mathbf{X}$. Note that the eigenvalues are strictly positive, because the columns of $\mathbf{X}$ are linearly independent. Let $\mathbf{e}_1, ..., \mathbf{e}_p$ be the standardized eigenvectors associated with $\lambda_1, ..., \lambda_p$. The vector $\mathbf{a}$ with $|\mathbf{a}| = 1$ that maximizes $|\mathbf{Xa}|$ is $\mathbf{a} = \mathbf{e}_1$.

Proof:
We first compute $|\mathbf{Xe}_1|^2 = \mathbf{e}_1^T\mathbf{X}^T\mathbf{Xe}_1 = \mathbf{e}_1^T\lambda_1\mathbf{e}_1 = \lambda_1\mathbf{e}_1^T\mathbf{e}_1 = \lambda_1$.

Now let $\mathbf{a}$ be any vector with length $|\mathbf{a}| = 1$. We show that $|\mathbf{Xa}|^2 \leq \lambda_1$. Let $\mathbf{\Gamma\Lambda\Gamma}^T$ be the eigendecomposition of $\mathbf{X}^T\mathbf{X}$. Let $\mathbf{b} = \mathbf{\Gamma}^T\mathbf{a}$. Then $|\mathbf{b}|^2 = \mathbf{b}^T\mathbf{b} = \mathbf{a}^T\mathbf{\Gamma\Gamma}^T\mathbf{a} = \mathbf{a}^T\mathbf{a} = |\mathbf{a}|^2 = 1$.

Now $|\mathbf{Xa}|^2 = \mathbf{a}^T\mathbf{X}^T\mathbf{Xa} = \mathbf{a}^T\mathbf{\Gamma\Lambda\Gamma}^T\mathbf{a} = \mathbf{b}^T\mathbf{\Lambda b} = \sum_{i=1}^p \lambda_i b_i^2$.
Since λ_1 is the largest eigenvalue, $\sum_{i=1}^p \lambda_i b_i^2 \leq \lambda_1 \sum_{i=1}^p b_i^2 = \lambda_1|\mathbf{b}|^2 = \lambda_1$.

We now show that if $\lambda_1 > \lambda_2$, then the only linear combination of the columns of $\mathbf{X}$ with squared length λ_1 is $\mathbf{Xe}_1$.

Let $\mathbf{a}$ be a vector with $|\mathbf{a}| = 1$ such that $|\mathbf{Xa}|^2 = \lambda_1$, and let $\mathbf{b} = \mathbf{\Gamma}^T\mathbf{a}$. Then, as shown above, $|\mathbf{b}| = 1$, and $|\mathbf{Xa}|^2 = \mathbf{b}^T\Lambda\mathbf{b} = \sum_{i=1}^p \lambda_i b_i^2 = \lambda_1$, so $b_1 = 1$ and $b_i = 0$ for $i \neq 1$. Therefore $\mathbf{\Gamma}^T\mathbf{a} = (1, 0, ..., 0)^T$, so $\mathbf{\Gamma}(1, 0, ..., 0)^T = \mathbf{a}$. But $\mathbf{\Gamma}(1, 0, ..., 0)^T$ is the first column of $\mathbf{\Gamma}$ which is $\mathbf{e}_1$, so $\mathbf{a} = \mathbf{e}_1$.

In general, if $\lambda_1 = \lambda_2 = \cdots = \lambda_k$, $\mathbf{a}$ may be any linear combination of $\mathbf{e}_1, ..., \mathbf{e}_k$.

Definition 6.2. Let $\mathbf{X}$ be an $n \times p$ centered matrix of full rank. The first **principal component** is the linear combination $\mathbf{z}_1 = \mathbf{X}\mathbf{a}_1$ whose length is maximum, subject to the constraint $|\mathbf{a}_1| = 1$. The second principal component is the linear combination $\mathbf{z}_2 = \mathbf{X}\mathbf{a}_2$ whose length is maximum, subject to the constraints $|\mathbf{a}_2| = 1$ and $\mathbf{z}_2 \perp \mathbf{z}_1$. In general, let $\mathbf{z}_1, ..., \mathbf{z}_k$ be the first k principal components, with $\mathbf{z}_i = \mathbf{X}\mathbf{a}_i$. The $k+1$st principal component is the linear combination $\mathbf{z}_{k+1} = \mathbf{X}\mathbf{a}_{k+1}$ whose length is maximum, subject to the constraints $|\mathbf{a}_{k+1}| = 1$ and $\mathbf{z}_{k+1} \perp \mathbf{z}_i$, for $i = 1, ..., k$.

Note that the total number of principal components is p, the number of columns of $\mathbf{X}$. We will now show how to find the principal components. First, a lemma.

Lemma:
Let $\mathbf{X}$ be an $n \times p$ centered matrix of full rank. Let $\mathbf{a}$ be a vector and let $\mathbf{e}$ be an eigenvector of $\mathbf{X}^T\mathbf{X}$ with eigenvalue $\lambda > 0$. If $\mathbf{X}\mathbf{a} \perp \mathbf{X}\mathbf{e} = 0$, then $\mathbf{a} \perp \mathbf{e}$, i.e., $\mathbf{a}^T\mathbf{e} = 0$.

Proof: $(\mathbf{X}\mathbf{a})^T(\mathbf{X}\mathbf{e}) = \mathbf{a}^T\mathbf{X}^T\mathbf{X}\mathbf{e} = \mathbf{a}^T\lambda\mathbf{e} = \lambda\mathbf{a}^T\mathbf{e}$. It follows that if $(\mathbf{X}\mathbf{a})^T(\mathbf{X}\mathbf{e}) = 0$, then $\mathbf{a}^T\mathbf{e} = 0$.

The following theorem shows how to find the principal components.

Theorem 6.2. Let $\mathbf{X}$ be an $n \times p$ centered matrix of full rank. Let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p > 0$ be the eigenvalues of $\mathbf{X}^T\mathbf{X}$. Note that all the eigenvalues are strictly positive. Let $\mathbf{e}_1, ..., \mathbf{e}_p$ be the eigenvectors associated with $\lambda_1, ..., \lambda_p$. Then for $k = 1, ..., p$, the kth principal component is $\mathbf{z}_k = \mathbf{X}\mathbf{e}_k$.

Proof: We have shown that the first principal component is $\mathbf{X}\mathbf{e}_1$. Now assume the first k principal components are $\mathbf{X}\mathbf{e}_1, ..., \mathbf{X}\mathbf{e}_k$. Let $\mathbf{a}$ be the vector that maximizes $\mathbf{X}\mathbf{a}$, subject to $|\mathbf{a}| = 1$ and $\mathbf{X}\mathbf{a} \perp \mathbf{X}\mathbf{e}_i$ for $i = 1, ..., k$. We show:
1) $|\mathbf{X}\mathbf{e}_{k+1}|^2 = \lambda_{k+1}$.
2) $|\mathbf{X}\mathbf{a}|^2 \leq \lambda_{k+1}$ for all $\mathbf{a}$.

1) $|\mathbf{X}\mathbf{e}_{k+1}|^2 = \mathbf{e}_{k+1}^T\mathbf{X}^T\mathbf{X}\mathbf{e}_{k+1} = \mathbf{e}_{k+1}\lambda_{k+1}\mathbf{e}_{k+1} = \lambda_{k+1}\mathbf{e}_{k+1}^T\mathbf{e}_{k+1} = \lambda_{k+1}$.

2) Since $\mathbf{X}\mathbf{a} \perp \mathbf{X}\mathbf{e}_i = 0$ for $i = 1, ..., k$, $\mathbf{a}^T\mathbf{e}_i = 0$ for $i = 1, ..., k$. Now let $\mathbf{\Gamma}\mathbf{\Lambda}\mathbf{\Gamma}^T$ be the eigendecomposition of $\mathbf{X}^T\mathbf{X}$ and let $\mathbf{b} = \mathbf{\Gamma}^T\mathbf{a} = (\mathbf{e}_1^T\mathbf{a}, ..., \mathbf{e}_k^T\mathbf{a}, \mathbf{e}_{k+1}^T\mathbf{a}, ..., \mathbf{e}_p^T\mathbf{a})^T$. It follows

that $b_1 = \cdots = b_k = 0$. We proved earlier that $|\mathbf{b}| = 1$. Now

$$\begin{aligned}
|\mathbf{Xa}|^2 &= \mathbf{a}^T\mathbf{X}^T\mathbf{Xa} \\
&= \mathbf{a}^T\mathbf{\Gamma\Lambda\Gamma}^T\mathbf{a} \\
&= \mathbf{b}^T\mathbf{\Lambda b} \\
&= \lambda_1 b_1^2 + \cdots + \lambda_k b_k^2 + \lambda_{k+1} b_{k+1}^2 + \cdots + \lambda_p b_p^2 \\
&= \lambda_{k+1} b_{k+1}^2 + \cdots + \lambda_p b_p^2 \\
&\leq \lambda_{k+1} \sum_{i=k+1}^{p} b_i \\
&\leq \lambda_{k+1}
\end{aligned}$$

We now see that for an $n \times p$ centered matrix of full rank $\mathbf{X}$, there are p principal components $\mathbf{z}_1 = \mathbf{Xe}_1$, ..., $\mathbf{z}_p = \mathbf{Xe}_p$. The principal components are linear combinations of the columns of $\mathbf{X}$ with the following properties:
1) $|\mathbf{z}_i|^2 = \lambda_i$.
2) $\mathbf{z}_i \perp \mathbf{z}_j$ for $i \neq j$.

Terminology:
Let $\mathbf{x}_1, ..., \mathbf{x}_p$ be the columns of $\mathbf{X}$. The ith principal component is $\mathbf{z}_i = \mathbf{Xe}_i = e_{i1}\mathbf{x}_1 + \cdots + e_{ip}\mathbf{x}_p$. The components $e_{i1}, ..., e_{ip}$ are called the **coefficients**, or **loadings**, of the principal component. In particular, e_{ik} is the loading of $\mathbf{z}_i$ on $\mathbf{x}_k$.

Definition 6.3. Let $\mathbf{X}$ be an $n \times p$ matrix. The total variance of $\mathbf{X}$ is $\sum_{i=1}^{p} |\mathbf{x}_i|^2$, the sum of the squared lengths of the columns of $\mathbf{X}$. Recall that because the columns are centered, the squared lengths are proportional to the variances.

The total variance of a centered matrix $\mathbf{X}$ of full rank turns out the be the sum of the eigenvalues of $\mathbf{X}^T\mathbf{X}$, as the next theorem shows.

Theorem 6.3. Let $\mathbf{X}$ be an $n \times p$ centered matrix of full rank, let $\mathbf{x}_1, ..., \mathbf{x}_p$ be the columns of $\mathbf{X}$, and let $\lambda_1, ..., \lambda_p$ be the eigenvalues of $\mathbf{X}^T\mathbf{X}$. Then $\sum_{i=1}^{p} \lambda_i = \sum_{i=1}^{p} |\mathbf{x}_i|^2$.

Proof: The ith diagonal element of $\mathbf{X}^T\mathbf{X}$ is $\mathbf{x}_i^T\mathbf{x}_i = |\mathbf{x}_i|^2$. Therefore $\sum_{i=1}^{p} |\mathbf{x}_i|^2 = tr(\mathbf{X}^T\mathbf{X}) = tr(\mathbf{\Gamma\Lambda\Gamma}^T) = tr(\mathbf{\Lambda\Gamma\Gamma}^T) = tr(\mathbf{\Lambda}) = \sum_{i=1}^{p} \lambda_i$.

Theorem 6.3 provides the basis for the method for choosing a subset of the principal components to replace $\mathbf{X}$. Let $\mathbf{Z}$ be the $n \times p$ matrix whose columns are $\mathbf{z}_1, ..., \mathbf{z}_p$, the principal components of $\mathbf{X}$, in decreasing order of the eigenvalues $\lambda_1, ..., \lambda_p$. The squared length of $\mathbf{z}_i$ is $\mathbf{z}_i^T \mathbf{z}_i = \lambda_i$, so the total variance of $\mathbf{Z}$ is the same as that of $\mathbf{X}$. We choose the first k principal components $\mathbf{z}_1, ..., \mathbf{z}_k$, where k is large enough so that total variance of $\mathbf{z}_1, ..., \mathbf{z}_k$ is a large fraction of the total variance of $\mathbf{X}$. The following definition makes this precise.

Definition 6.4. Let $\mathbf{X}$ be an $n \times p$ centered matrix of full rank and let $\lambda_1 \geq \cdots \geq \lambda_p$ be the eigenvalues of $\mathbf{X}^T\mathbf{X}$ corresponding to eigenvectors $\mathbf{e}_1$, ..., $\mathbf{e}_p$. Let $\mathbf{z}_1 = \mathbf{X}\mathbf{e}_1$, ..., $\mathbf{z}_p = \mathbf{X}\mathbf{e}_p$ be the principal components of $\mathbf{X}$. The proportion of variance explained by $\mathbf{z}_i$ is $\dfrac{\lambda_i}{\sum_{i=1}^{p} \lambda_i}$. The proportion of variance explained by $\mathbf{z}_1, ..., \mathbf{z}_k$ is $\dfrac{\sum_{j=1}^{k} \lambda_j}{\sum_{i=1}^{p} \lambda_i}$.

Clearly, the first principal component explains more of the variance than the others, the second explains the second most, and so forth. It turns out that in many cases the first few principal components explain a large proportion of the variance. In these cases we can replace $\mathbf{x}_1, ..., \mathbf{x}_p$ with a much smaller set of variables $\mathbf{z}_1, ..., \mathbf{z}_k$ which explain most of the variance.

Example 6.1. Let $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ be a centered matrix with $\mathbf{X}^T\mathbf{X} = \begin{bmatrix} 14 & -9 & 7 \\ -9 & 30 & 1 \\ 7 & 1 & 14 \end{bmatrix}$.

1. Find the principal components.
2. How much of the variance is explained by the first principal component? By the first two?

Solution: We use R to find the eigenvalues and eigenvectors.

```
> xtx = matrix(c(14, -9, 7, -9, 30, 1, 7, 1, 14), ncol=3)

> xtx
     [,1] [,2] [,3]
[1,]   14   -9    7
[2,]   -9   30    1
[3,]    7    1   14
```

```
> eigen(xtx)
$values
[1] 34.249965 18.907364  4.842671

$vectors
           [,1]      [,2]       [,3]
[1,]  0.4342000 0.5068504  0.7446966
[2,] -0.8945679 0.3398130  0.2903022
[3,]  0.1059178 0.7922309 -0.6009589
```

The total variance is 14 + 30 + 14 = 58. (This is also the sum of the eigenvalues.) We summarize the results:

Principal Component	Variance	Proportion Explained	Cumulative P. E.
$\mathbf{z}_1 = 0.434\mathbf{x}_1 - 0.895\mathbf{x}_2 + 0.106\mathbf{x}_3$	34.25	0.5905	0.5905
$\mathbf{z}_2 = 0.507\mathbf{x}_1 + 0.340\mathbf{x}_2 + 0.792\mathbf{x}_3$	18.91	0.3260	0.9165
$\mathbf{z}_3 = 0.745\mathbf{x}_1 + 0.290\mathbf{x}_2 - 0.601\mathbf{x}_3$	4.843	0.0835	1.0000

The first principal component explains 59.05% of the variance. The first two principal components explain 91.65%. This suggests that we might replace $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ with $\mathbf{z}_1, \mathbf{z}_2$.

6.3 Principal Components Based on Standardized Variables

If a column of $\mathbf{X}$ is multiplied by a constant, the principal components will change. This is unfortunate, because it means that the principal components depend on the units chosen for the columns of $\mathbf{X}$. A way around this is to divide each element of $\mathbf{X}$ by the standard deviation of its column, so that each of the columns has variance 1. This process is referred to as **standardizing** the columns of $\mathbf{X}$.

For each column $\mathbf{x}_i$, let $s_i = \sqrt{\frac{\sum_{j=1}^n x_{ji}^2}{n-1}}$, Note that since the columns have mean 0, $s_i = \sqrt{\frac{\sum_{j=1}^n (x_{ji} - \bar{x}_{\cdot i})^2}{n-1}}$, which is essentially the standard deviation of the elements in $\mathbf{x}_i$.

We can compute principal components from the standardized matrix with columns $\frac{\mathbf{x}_i}{s_i}$. The principal components computed from the standardized matrix are often quite different from the ones computed from the original matrix.

Example 6.2. Refer to Example 6.1. Assume the dimensions of the matrix $\mathbf{X}$ are 10×3. Find the principal components for the standardized matrix.

Solution: We aren't given the original matrix $\mathbf{X}$, but we know that $\mathbf{X}^T\mathbf{X} = \begin{bmatrix} 14 & -9 & 7 \\ -9 & 30 & 1 \\ 7 & 1 & 14 \end{bmatrix}$.

The standard deviations of the columns are $s_1 = \sqrt{14/9}$, $s_2 = \sqrt{30/9}$, $s_3 = \sqrt{14/9}$,

The standardized variables are $\mathbf{w}_1 = \dfrac{\mathbf{x}_1}{s_1}$, $\mathbf{w}_2 = \dfrac{\mathbf{x}_2}{s_2}$, $\mathbf{w}_3 = \dfrac{\mathbf{x}_3}{s_3}$.

We use R to compute $\mathbf{W}^T\mathbf{W}$.

```
> s = c(sqrt(14/9), sqrt(30/9), sqrt(14/9))
> wtw = diag(1/s)%*%xtx%*%diag(1/s)

          [,1]      [,2]     [,3]
[1,]  9.000000 -3.952395 4.500000
[2,] -3.952395  9.000000 0.439155
[3,]  4.500000  0.439155 9.000000

> eigen(wtw)
$values
[1] 14.775781  9.435444  2.788775

$vectors
           [,1]         [,2]       [,3]
[1,]  0.7198003 -0.009561758  0.6941153
[2,] -0.4525387  0.751767111  0.4796404
[3,]  0.5263993  0.659359373 -0.5367951
```

The total variance is 30. We summarize the results:

Principal Component	Variance	Proportion Explained	Cumulative P. E.
$\mathbf{z}_1 = 0.720\mathbf{w}_1 - 0.453\mathbf{w}_2 + 0.106\mathbf{w}_3$	14.776	0.5473	0.5473
$\mathbf{z}_2 = -0.010\mathbf{w}_1 + 0.752\mathbf{w}_2 + 0.659\mathbf{w}_3$	9.4354	0.3493	0.8966
$\mathbf{z}_3 = 0.694\mathbf{w}_1 + 0.480\mathbf{w}_2 - 0.537\mathbf{w}_3$	3.7888	0.1033	1.0000

The first principal component explains 54.73% of the variation. The first two principal components explain 89.66%. This suggests that we might replace $\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3$ with $\mathbf{z}_1, \mathbf{z}_2$.

Examples 6.1 and 6.2 show that the principal components calculated from the standardized variables are generally different from those calculated from the original ones. In some cases

they can be very different. This is particularly likely to occur when one of the variables has a substantially larger variance than the others. Example 6.3 provides an illustration.

Example 6.3. Let $\mathbf{X}$ be a centered centered matrix with $\mathbf{X}^T\mathbf{X} = \begin{bmatrix} 100.0 & 5.0 & 4.0 \\ 5.0 & 1.0 & 0.5 \\ 4.0 & 0.5 & 1.0 \end{bmatrix}$. Assume $\mathbf{X}$ has 20 columns.

1. Find the principal components $\mathbf{z}_1$, $\mathbf{z}_2$, and $\mathbf{z}_3$ and the proportion of variance explained by each.

2. Find the principal components of the standardized variables and the proportion of variance explained by each.

Solution: We use R to compute the eigenvalues and eigenvectors of $\mathbf{X}^T\mathbf{X}$.

```
> xtx = matrix(c(100, 5, 4, 5, 1, 0.5, 4, 0.5, 1), ncol=3)
> xtx
     [,1] [,2] [,3]
[1,]  100  5.0  4.0
[2,]    5  1.0  0.5
[3,]    4  0.5  1.0
> eigen(xtx)
$values
[1] 100.4144490   1.0939760   0.4915749

$vectors
           [,1]        [,2]        [,3]
[1,] 0.99791182  0.06348893  0.01188045
[2,] 0.05039269 -0.65020953 -0.75808189
[3,] 0.04040503 -0.75709757  0.65205116
```

The total variance is 102. We summarize the results:

Principal Component	Variance	Proportion Explained	Cumulative P. E.
$\mathbf{z}_1 = 0.998\mathbf{x}_1 + 0.050\mathbf{x}_2 + 0.040\mathbf{x}_3$	100.414	0.9844	0.9844
$\mathbf{z}_2 = 0.063\mathbf{x}_1 - 0.650\mathbf{x}_2 - 0.758\mathbf{x}_3$	1.094	0.0107	0.9952
$\mathbf{z}_3 = 0.012\mathbf{x}_1 - 0.758\mathbf{x}_2 - 0.652\mathbf{x}_3$	0.492	0.0048	1.0000

The first principal component is almost equal to the first column $\mathbf{x}_1$, and it explains more than 98% of the variation. The reason the first variable dominates is that it has by far the largest variance of the three.

Now we find the principal components of the standardized matrix. Assume the matrix has 20 columns.

```
> s = c(sqrt(100/19), sqrt(1/19), sqrt(1/19))
> wtw = diag(1/s)%*%xtx%*%diag(1/s)

> wtw
     [,1] [,2] [,3]
[1,] 19.0  9.5  7.6
[2,]  9.5 19.0  9.5
[3,]  7.6  9.5 19.0

> eigen(wtw)
$values
[1] 36.762092 11.400000  8.837908

$vectors
           [,1]          [,2]       [,3]
[1,] 0.5639516  7.071068e-01  0.4265661
[2,] 0.6032555  1.110223e-16 -0.7975480
[3,] 0.5639516 -7.071068e-01  0.4265661
```

The total variance is 60. We summarize the results:

Principal Component	Variance	Proportion Explained	Cumulative P. E.
$\mathbf{z}_1 = 0.564\mathbf{w}_1 + 0.603\mathbf{w}_2 + 0.564\mathbf{w}_3$	36.762	0.6450	0.6450
$\mathbf{z}_2 = 0.707\mathbf{w}_1 + 0.000\mathbf{w}_2 + 0.707\mathbf{w}_3$	11.400	0.2000	0.8450
$\mathbf{z}_3 = 0.427\mathbf{w}_1 - 0.798\mathbf{w}_2 + 0.427\mathbf{w}_3$	8.8379	0.1550	1.0000

Now the first principal component weights the three variables approximately equally. It also explains only 64.5% of the variation.

The preceding example suggests that principal components computed from standardized matrix are more useful, especially when a few of the columns have much greater lengths than the others. Another advantage of the standardized columns is that they are unitless, whereas the principal components computed from the original matrix are linear combinations of columns with differing units.

Example 6.4. Values for the following variables were collected on each of 61 census tracts in the Madison, Wisconsin, area. The variables were

$\mathbf{x}_1$: Total population (1000s)

$\mathbf{x}_2$: Percent with professional degrees

$\mathbf{x}_3$: Percent employed

$\mathbf{x}_4$: Percent employed by government

$\mathbf{x}_5$: Median home value ($100,000s)

The data are available in the file

`http://www.mines.edu/~wnavidi/math437537/census.txt`

Find the principal components and determine how many components are needed to adequately summarize the five variables.

Solution: We use R to read the data, center the matrix, and compute $\mathbf{X}^T\mathbf{X}$. Note that we are not standardizing the variables. We note that the `cov` command in R returns the matrix $\mathbf{X}^T\mathbf{X}/(n-1)$ where $\mathbf{X}$ is the matrix obtained after centering the columns (in other words, it returns the covariance matrix).

```
> census = read.table("http://www.mines.edu/~wnavidi/math437537/census.txt",
header = TRUE)

> n = nrow(census)
> xtx = (n-1)*cov(census)
> xtx
            X1        X2           X3          X4        X5
X1  203.813941 -66.12836   258.333290  -124.69711  1.632234
X2  -66.128362 580.36648   -90.794177   657.19393 72.183820
X3  258.333290 -90.79418  3337.554698 -1736.24785 -2.613544
X4 -124.697115 657.19393 -1736.247852  5343.99672 57.437984
X5    1.632234  72.18382    -2.613544    57.43798 19.117505

> eigen(xtx)
$values
[1] 6420.915212 2380.328150  502.251959  172.072438    9.281585

$vectors
             [,1]        [,2]        [,3]        [,4]         [,5]
[1,]  0.038887287 -0.07114494 -0.18789258  0.97713524 -0.057699864
[2,] -0.105321969 -0.12975236  0.96099580  0.17135181 -0.138554092
[3,]  0.492363944 -0.86438807 -0.04579737 -0.09104368  0.004966048
[4,] -0.863069865 -0.48033178 -0.15318538 -0.02968577  0.006691800
[5,] -0.009122262 -0.01474342  0.12498114  0.08170118  0.988637470
```

```
#Now compute the proportion and cumulative proportion of variance explained for
#each principal component.

> totvar = sum(eigen(xtx)$values)
> totvar
[1] 9484.849

> prop = eigen(s)$values/totvar
> prop
[1] 0.6769654402 0.2509610921 0.0529530772 0.0181418208 0.0009785696

> cumprop = cumsum(prop)
> cumprop
[1] 0.6769654 0.9279265 0.9808796 0.9990214 1.0000000
```

Following is a table of principal components:

Principal Component	Variance	Proportion Explained	Cumulative P. E.
$\mathbf{z}_1 = 0.0389\mathbf{x}_1 - 0.1053\mathbf{x}_2 + 0.4924\mathbf{x}_3 - 0.8631\mathbf{x}_4 - 0.0091\mathbf{x}_5$	107.02	0.677	0.677
$\mathbf{z}_2 = -0.0711\mathbf{x}_1 - 0.1298\mathbf{x}_2 - 0.8644\mathbf{x}_3 - 0.4803\mathbf{x}_4 - 0.0147\mathbf{x}_5$	39.67	0.251	0.928
$\mathbf{z}_3 = -0.1879\mathbf{x}_1 + 0.9610\mathbf{x}_2 - 0.0458\mathbf{x}_3 - 0.1532\mathbf{x}_4 + 0.1250\mathbf{x}_5$	8.37	0.053	0.981
$\mathbf{z}_4 = 0.9771\mathbf{x}_1 + 0.1714\mathbf{x}_2 - 0.0910\mathbf{x}_3 - 0.0297\mathbf{x}_4 + 0.0817\mathbf{x}_5$	2.87	0.018	0.999
$\mathbf{z}_5 = -0.0577\mathbf{x}_1 - 0.1386\mathbf{x}_2 + 0.0050\mathbf{x}_3 + 0.0067\mathbf{x}_4 + 0.9886\mathbf{x}_5$	0.15	0.001	1.000

The first two principal component explain 92.8% of the variance. It might seem reasonable to replace the five original variables with the first two principal components. However, we did not standardize the variables, so this conclusion might not be warranted.

Now we will standardize the variables, to eliminate the problem of differing units. A look at the matrix $\mathbf{X}^T\mathbf{X}$ in Example 6.4 shows that the lengths of $\mathbf{x}_3$ and $\mathbf{x}_4$ are noticeably larger than the rest. This may be the reason that these two variables are prominent in the first two principal components. Standardization may make a substantial difference in this case.

Example 6.5. Refer to Example 6.4. Find the principal components from the standardized variables. Is it reasonable to summarize the data with the first two principal components?

Solution: Let $\mathbf{W}$ be the standardized version of $\mathbf{X}$. We note that the `cor` command in R returns the matrix $\mathbf{W}^T\mathbf{W}/(n-1)$ (in other words, it returns the correlation matrix of $\mathbf{X}$).

```
> wtw = (n-1)*cor(census)
> wtw
           X1         X2          X3         X4         X5
X1  60.000000 -11.536416  18.7931892  -7.168984  1.5689216
X2 -11.536416  60.000000  -3.9142079  22.390333 41.1172769
X3  18.793189  -3.914208  60.0000000 -24.666963 -0.6207994
X4  -7.168984  22.390333 -24.6669631  60.000000 10.7820599
X5   1.568922  41.117277  -0.6207994  10.782060 60.0000000

> eigen(wtw)
$values
[1] 119.51510  82.05160  51.84944  32.10366  14.48020

$vectors
           [,1]       [,2]        [,3]       [,4]       [,5]
[1,]  0.2625829  0.4629936  0.78390268  0.2169291  0.2347882
[2,] -0.5933541  0.3256442 -0.16407255 -0.1446471  0.7028828
[3,]  0.3256978  0.6051419 -0.22487455 -0.6628689 -0.1943206
[4,] -0.4792022 -0.2524850  0.55070086 -0.5716730 -0.2766497
[5,] -0.4932213  0.4996473 -0.06882436  0.4072024 -0.5801162

#Now compute the proportion and cumulative proportion of variance explained for
#each principal component.

> prop = eigen(wtw)$values/300
> prop
[1] 0.39838366 0.27350533 0.17283146 0.10701220 0.04826735

> cumsum(prop)
[1] 0.3983837 0.6718890 0.8447204 0.9517327 1.0000000
```

Following is a table of principal components:

Principal Component	Variance	Proportion Explained	Cumulative P. E.
$\mathbf{z}_1 = 0.2626\mathbf{x}_1 - 0.5934\mathbf{x}_2 + 0.3257\mathbf{x}_3 - 0.4792\mathbf{x}_4 - 0.4932\mathbf{x}_5$	119.5	0.398	0.398
$\mathbf{z}_2 = 0.4630\mathbf{x}_1 + 0.3256\mathbf{x}_2 + 0.6051\mathbf{x}_3 - 0.2525\mathbf{x}_4 + 0.4996\mathbf{x}_5$	82.05	0.274	0.672
$\mathbf{z}_3 = 0.7839\mathbf{x}_1 - 0.1641\mathbf{x}_2 - 0.2249\mathbf{x}_3 + 0.5507\mathbf{x}_4 - 0.0688\mathbf{x}_5$	51.85	0.173	0.845
$\mathbf{z}_4 = 0.2169\mathbf{x}_1 - 0.1446\mathbf{x}_2 - 0.6629\mathbf{x}_3 - 0.5717\mathbf{x}_4 + 0.4072\mathbf{x}_5$	32.10	0.107	0.952
$\mathbf{z}_5 = 0.2348\mathbf{x}_1 + 0.7029\mathbf{x}_2 - 0.1943\mathbf{x}_3 - 0.2766\mathbf{x}_4 - 0.5802\mathbf{x}_5$	14.48	0.048	1.000

The first two principal components explain only 67.2% of the variation. We need three or four principal components to explain most of the variation.

Should We Compute Principal Components from the Original Variables or from the Standardized Variables?

If the columns of $\mathbf{X}$ have differing units, the principal components computed from the original variables are linear combinations of vectors with differing units, and thus hard to interpret. In addition, variables with larger variances tend to dominate the larger principal components. With standardized variables, the principal components are linear combinations of unitless vectors, and thus unitless. In addition, the variables are all scaled to have variance 1, so differences in the variances have no effect. As a rule of thumb, if variables have different units of measurement (e.g. kilograms, liters, percentage) it is better to standardize them. If all the variables are measured in the same units, then using the original variables can be justified, but it still may be better to standardize them.

6.4 Principal Components Regression (PCR)

Let $\mathbf{1}$ be the $n \times 1$ vector all of whose components are 1. Let $\mathbf{x}_1, ..., \mathbf{x}_p$ be $n \times 1$ vectors of constants. We will assume that the variables are standardized, so the components of each $\mathbf{x}_i$ have mean 0 and standard deviation 1. In particular $\mathbf{1}^T\mathbf{x}_i = 0$ for all i. We will consider the multiple regression model with intercept

$$\mathbf{Y} = \beta_0\mathbf{1} + \beta_1\mathbf{x}_1 + \cdots + \beta_p\mathbf{x}_p = \boldsymbol{\varepsilon}$$

Let $\mathbf{X} = |\, \mathbf{x}_1 \ \mathbf{x}_2 \ \cdots \ \mathbf{x}_p \,|$ be the matrix whose columns are $\mathbf{x}_1$, ..., $\mathbf{x}_p$. Note that we have omitted the intercept.

Before we begin PCR, we will estimate β_0. The average of the components of $\mathbf{Y}$ is

$$\begin{aligned}
\bar{\mathbf{Y}} &= \frac{1}{n}\mathbf{1}^T\mathbf{Y} \\
&= \beta_0\frac{1}{n}\mathbf{1}^T\mathbf{1} + \beta_1\frac{1}{n}\mathbf{1}^T\mathbf{x}_1 + \cdots \beta_p\frac{1}{n}\mathbf{1}^T\mathbf{x}_p + \frac{1}{n}\mathbf{1}^T\boldsymbol{\varepsilon} \\
&= \frac{1}{n}n\beta_0 + \bar{\boldsymbol{\varepsilon}} \\
&= \beta_0 + \bar{\boldsymbol{\varepsilon}}
\end{aligned}$$

The least-squares estimator of β_0 is $\hat{\beta}_0 = \bar{\mathbf{Y}}$.

The idea behind PCR is to replace the columns $\mathbf{x}_1, ..., \mathbf{x}_p$ with the principal components of $\mathbf{X}$. The principal components with smaller eigenvalues represent directions of less variation in $\mathbf{X}$. The idea is that they can be dropped from the model, thereby reducing the dimension

presumably without seriously affecting model performance. We will use the standardized variables to compute the eigenvalues and eigenvectors.

Procedure

Let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p$ be the eigenvalues of $\mathbf{X}^T\mathbf{X}$, with corresponding eigenvectors $\mathbf{e}_1$, ..., $\mathbf{e}_p$ (recall that the columns of $\mathbf{X}$ have been standardized). The principal components are linear combinations of the columns of $\mathbf{X}$ whose coefficients are the components of the eigenvectors. Specifically, the first principal component is $\mathbf{z}_1 = e_{11}\mathbf{x}_1 + \cdots + e_{1p}\mathbf{x}_p = \mathbf{X}\mathbf{e}_1$ and the others are $\mathbf{z}_2 = \mathbf{X}\mathbf{e}_2$, ..., $\mathbf{z}_p = \mathbf{X}\mathbf{e}_p$.

Let $\mathbf{Z}$ be the $n \times p$ matrix whose ith column is $\mathbf{z}_i$. Let $\mathbf{\Gamma}$ be the matrix whose columns are $\mathbf{e}_1, ..., \mathbf{e}_p$, the eigenvectors of $\mathbf{X}^T\mathbf{X}$. Then

$$\mathbf{Z} = \mathbf{X}\mathbf{\Gamma}$$

Let $\boldsymbol{\gamma} = \mathbf{\Gamma}^T\boldsymbol{\beta}$. Then

$$\mathbf{Z}\boldsymbol{\gamma} = \mathbf{X}\mathbf{\Gamma}\mathbf{\Gamma}^T\boldsymbol{\beta} = \mathbf{X}\boldsymbol{\beta}$$

So we can rewrite the model as

$$\mathbf{Y} = \beta_0\mathbf{1} + \mathbf{Z}\boldsymbol{\gamma} + \boldsymbol{\varepsilon}$$

We can use this model to compute the least-squares estimate $\hat{\boldsymbol{\gamma}}$ of $\boldsymbol{\gamma}$. Then the least-squares estimate of $\boldsymbol{\beta}$ is $\hat{\boldsymbol{\beta}} = \mathbf{\Gamma}\hat{\boldsymbol{\gamma}}$.

Following are the steps in principal components regression:

1. Choose a number k of principal components to include.
2. Compute the least-squares estimate $\hat{\boldsymbol{\gamma}} = (\mathbf{Z}^T\mathbf{Z})^{-1}\mathbf{Z}^T\mathbf{Y}$.
3. Construct the reduced estimator $\hat{\boldsymbol{\gamma}}_k$ by replacing the last $p - k + 1$ components of $\hat{\boldsymbol{\gamma}}$ with zeros. So $\hat{\boldsymbol{\gamma}}_k = (\hat{\gamma}_1, ..., \hat{\gamma}_k, 0, ..., 0)^T$.
4. Compute the reduced estimator $\hat{\boldsymbol{\beta}}_k = \mathbf{\Gamma}\hat{\boldsymbol{\gamma}}_k$.

Example 6.6. Given the standardized design matrix

$$\mathbf{X} = \begin{vmatrix} 0.01 & 0.39 & 0.31 & 2.17 & -1.17 & 0.22 & -0.64 & 0.28 & 0.01 & 2.35 \\ -2.26 & -2.21 & 1.18 & 0.62 & 0.73 & -0.52 & 0.48 & 1.53 & 0.27 & -0.22 \\ 0.25 & 1.91 & -2.20 & 0.29 & -0.97 & 0.75 & -0.51 & 0.61 & -1.98 & 1.57 \\ -1.29 & 1.52 & -1.70 & 1.49 & 1.45 & 2.77 & -1.66 & 2.64 & 0.09 & 0.87 \\ -0.48 & -0.57 & 0.93 & 0.22 & -0.08 & -0.65 & 0.64 & -0.05 & 0.01 & 0.39 \\ 0.01 & 0.73 & -0.45 & 0.22 & 0.73 & 1.22 & -1.10 & 0.21 & 0.23 & 0.39 \\ 1.22 & 0.05 & -0.32 & 1.56 & 0.98 & 1.47 & -1.83 & 1.26 & 1.71 & 0.82 \\ -0.88 & -0.86 & -0.07 & -1.26 & 1.96 & 0.67 & -0.51 & 0.94 & 1.50 & -1.88 \\ 0.57 & 0.61 & 1.81 & -0.25 & -1.66 & -1.42 & 1.40 & -0.64 & -1.07 & -0.70 \\ -0.23 & -1.31 & 1.06 & -0.12 & 0.19 & -0.56 & 0.91 & -1.23 & 0.88 & -0.44 \\ -1.61 & -0.74 & 0.68 & -0.72 & -0.36 & -1.21 & 1.73 & -0.64 & -0.78 & -0.44 \\ 0.17 & 0.39 & 0.06 & -1.53 & 0.38 & -0.05 & 0.18 & -0.71 & -0.35 & -1.09 \\ 0.74 & 0.10 & 0.06 & -0.05 & 0.36 & 0.30 & -0.41 & 0.48 & 1.06 & -0.48 \\ -0.15 & -0.57 & -0.07 & -0.65 & 0.53 & -0.22 & 0.18 & -0.57 & 0.34 & -0.62 \\ 0.17 & 0.16 & -0.95 & -0.65 & -1.22 & -1.02 & 1.70 & -0.57 & -1.51 & 0.21 \\ 0.01 & -1.42 & 0.81 & -0.79 & 0.78 & -0.56 & -0.48 & -0.51 & 1.35 & -1.44 \\ 1.55 & 0.78 & -1.32 & -1.59 & -0.28 & -0.24 & 0.22 & -1.30 & -0.46 & -0.31 \\ 0.41 & 0.61 & -0.32 & 0.22 & -1.14 & -0.44 & 0.64 & -0.51 & -1.22 & 0.78 \\ 1.87 & 0.44 & 0.56 & 0.69 & -1.32 & -0.58 & -0.28 & -0.44 & 0.12 & 0.39 \\ -0.07 & -0.01 & -0.07 & 0.15 & 0.11 & 0.07 & -0.67 & -0.77 & -0.20 & -0.14 \end{vmatrix}$$

and the vector of observations

$\mathbf{Y} = (25.68,\ 15.75,\ 3.98,\ 28.34,\ 10.72,\ 8.47,\ 27.85,\ 12.59,\ -0.94,\ 1.73,$
$2.00,\ -1.77,\ 17.03,\ 9.17,\ -5.81,\ 10.07,\ -3.06,\ 10.96,\ 9.99,\ -2.08)$

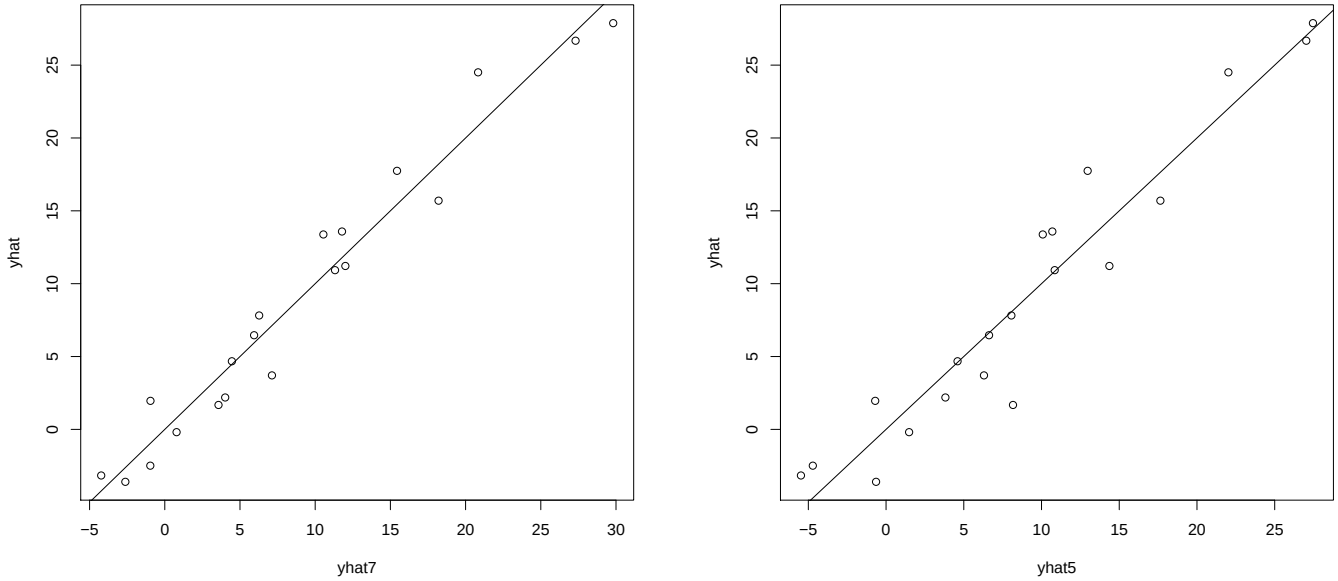
1. Fit the full model $\mathbf{Y} = \beta_0\mathbf{1} + \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$ and obtain the least-squares estimate $\hat{\beta}$.
2. Compute the eigenvectors of $\mathbf{X}^T\mathbf{X}$.
3. Construct the design matrix $\mathbf{Z}$ whose columns are the principal components of $\mathbf{X}$.
4. Fit the full model $\mathbf{Y} = \beta_0\mathbf{1} + \mathbf{Z}\boldsymbol{\gamma} + \boldsymbol{\varepsilon}$ and obtain the least-squares estimate $\hat{\gamma}$.
5. Construct the reduced estimator $\hat{\boldsymbol{\gamma}}_7$ using the first 7 principal components.
6. Compute the reduced estimator $\hat{\boldsymbol{\beta}}_7$.
7. Construct the reduced estimator $\hat{\boldsymbol{\gamma}}_4$ using the first 5 principal components.
8. Compute the reduced estimator $\hat{\boldsymbol{\beta}}_4$.
9. Compute the fitted values $\hat{\mathbf{Y}}$ from the fit of the full model, and from the two reduced models.
10. Plot the fitted values from the full model against those from each of the reduced models.

Solution: We use R:

```
> #Fit the full model
> out = lm(y~x)

> #The least squares estimate
> out$coeff
(Intercept)         xV1         xV2         xV3         xV4         xV5
  9.0266014  -3.0172388   8.3218932  -1.4180770   1.8159158   2.9313631
        xV6         xV7         xV8         xV9        xV10
-11.0087397  -0.3924024   6.0263866  11.3736199   5.3457469

> betahat = out$coeff

> #Eigenvalues and eigenvectors
> n = nrow(x)
> gamma = eigen((n-1)*cor(x))$vectors
> gamma
              [,1]        [,2]        [,3]        [,4]        [,5]        [,6]
 [1,]  0.005465411 -0.29344343  0.41317378  0.63531493 -0.10520505  0.39428263
 [2,] -0.238611379 -0.44270612  0.26378820 -0.07570341 -0.48738262 -0.34171035
 [3,]  0.306558894  0.23886383 -0.38655847  0.36622077 -0.49400395 -0.39973191
 [4,] -0.328862188 -0.10926908 -0.54997046  0.28557535  0.02042288 -0.04846924
 [5,] -0.218303419  0.48845349  0.21063874 -0.15169155  0.13997536 -0.13179000
 [6,] -0.486644423  0.07262031  0.15582351 -0.08132822 -0.03706791 -0.25712983
 [7,]  0.449378209 -0.10714775 -0.16599670 -0.27063549 -0.03019503  0.29344138
 [8,] -0.406290092  0.16191674 -0.24874783 -0.22601958 -0.51165462  0.61438215
 [9,] -0.127970954  0.47121774  0.07125022  0.46188615  0.12901116  0.13547277
[10,] -0.270255231 -0.38118531 -0.38033096  0.08764225  0.45455459 -0.01516186
              [,7]        [,8]        [,9]        [,10]
 [1,]  0.04733401  0.04153754  0.41056404 -0.021629929
 [2,]  0.29061441  0.08638403 -0.28033943  0.383332462
 [3,]  0.03813110  0.31884124  0.20264879 -0.131965390
 [4,]  0.07286281 -0.61684722  0.15872587  0.286564816
 [5,]  0.18440621  0.18344390  0.52458522  0.516303336
 [6,]  0.33908467 -0.14151801  0.20345775 -0.695066656
 [7,]  0.75473115 -0.10202136  0.13654692 -0.009381115
 [8,] -0.09947023  0.21429687 -0.01136206 -0.023420986
 [9,]  0.39092495  0.04654245 -0.59203361  0.051699184
[10,]  0.15880800  0.62966299  0.02690500 -0.007613619

> #We don't need to compute the eigenvalues.

> #Construct the matrix Z of principal components
> z = x%*%gamma

> #Fit the model using the matrix Z
```

```
> out1 = lm(y~z)

> #The least-squares estimate
> out1$coeff
(Intercept)          z1          z2          z3          z4          z5
   9.026601   -3.841691    1.636011   -3.256612    2.368965   -1.356828
         z6          z7          z8          z9         z10
   3.936985    3.561125    6.343786  -10.984718   13.537953

> g = out1$coeff[2:11]

> #Construct the reduced estimator using the first 7 principal components
> g7 = c(g[1:7], 0, 0, 0)
> b7 = gamma%*%g7

> #Compare the two estimates of beta
> cbind(betahat[2:11], b7)
 [1,]  -3.0172388  1.5220112
 [2,]   8.3218932 -0.4950952
 [3,]  -1.4180770  0.5718437
 [4,]   1.8159158  3.5931205
 [5,]   2.9313631  0.5403647
 [6,] -11.0087397  1.5337262
 [7,]  -0.3924024  1.8817299
 [8,]   6.0263866  4.8591962
 [9,]  11.3736199  3.8751408
[10,]   5.3457469  1.7499164

> #Construct the reduced estimator using the first 7 principal components
> g5 = c(g[1:5], 0,0,0,0,0)
> b5 = gamma%*%g5

> #Compare the three estimates of beta
> cbind(betahat[2:11], b7, b5)
             [,1]       [,2]       [,3]
 [1,]  -3.0172388  1.5220112 -0.1988360
 [2,]   8.3218932 -0.4950952 -0.1847009
 [3,]  -1.4180770  0.5718437  2.0097927
 [4,]   1.8159158  3.5931205  3.5244696
 [5,]   2.9313631  0.5403647  0.4025264
 [6,] -11.0087397  1.5337262  1.3385195
 [7,]  -0.3924024  1.8817299 -1.9612367
 [8,]   6.0263866  4.8591962  2.7946087
 [9,]  11.3736199  3.8751408  1.9496538
[10,]   5.3457469  1.7499164  1.2440732

> #Compute the fitted values for all three models
```

```
> yhat = 9.0266014 + x%*%betahat[2:11]
> yhat7 = 9.0266014 + x%*%b7
> yhat5 = 9.0266014 + x%*%b5

> #Plot fitted values from full model against those from reduced models
> plot(yhat7, yhat)
> plot(yhat5, yhat)
```



Left: Plot of $\hat{Y}$ vs. $\hat{Y}_{[7]}$. Right: Plot of $\hat{Y}$ vs. $\hat{Y}_{[5]}$. The lines are $y = x$. The fitted values from the seven-variable model are somewhat closer to those of the full model than those from the five-variable model are.

A Closer Look at PCR

Because the columns of $\mathbf{Z} = \mathbf{X}\boldsymbol{\Gamma}$ are the principal components of $\mathbf{X}$, the matrix $\mathbf{Z}$ is orthogonal. Now $\mathbf{X}^T\mathbf{X} = \boldsymbol{\Gamma}^T\boldsymbol{\Lambda}\boldsymbol{\Gamma}$, so

$$\mathbf{Z}^T\mathbf{Z} = \boldsymbol{\Gamma}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\Gamma} = \boldsymbol{\Gamma}^T\boldsymbol{\Gamma}\boldsymbol{\Lambda}\boldsymbol{\Gamma}^T\boldsymbol{\Gamma} = \boldsymbol{\Lambda}$$

which is diagonal. We conclude that

- $\mathbf{z}_i^T\mathbf{z}_j = 0$ for $i \neq j$
- $\mathbf{z}_i^T\mathbf{z}_i = \lambda_i$

Because $\mathbf{Z}$ is orthogonal, we can easily derive expressions for the components of the least-squares estimate $\hat{\gamma}_i$. What makes it easy is that $\mathbf{Z}^T\mathbf{Z}$ is a diagonal matrix:

$$\mathbf{Z}^T\mathbf{Z} = \begin{vmatrix} \mathbf{z}_1^T\mathbf{z}_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & \mathbf{z}_2^T\mathbf{z}_2 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0\cdots & 0 & 0 & \mathbf{z}_p^T\mathbf{z}_p \end{vmatrix} = \begin{vmatrix} \lambda_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & \lambda_2 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0\cdots & 0 & 0 & \lambda_p \end{vmatrix}$$

The least-squares estimate is

$$\hat{\boldsymbol{\gamma}} = (\mathbf{Z}^T\mathbf{Z})^{-1}\mathbf{Z}^T\mathbf{Y} = \begin{vmatrix} \frac{1}{\mathbf{z}_1^T\mathbf{z}_1} & 0 & 0 & \cdots & 0 & 0 \\ 0 & \frac{1}{\mathbf{z}_2^T\mathbf{z}_2} & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0\cdots & 0 & 0 & \frac{1}{\mathbf{z}_p^T\mathbf{z}_p} \end{vmatrix} \begin{vmatrix} \mathbf{z}_1^T\mathbf{Y} \\ \mathbf{z}_2^T\mathbf{Y} \\ \vdots \\ \mathbf{z}_p^T\mathbf{Y} \end{vmatrix} = \begin{vmatrix} \frac{\mathbf{z}_1^T\mathbf{Y}}{\mathbf{z}_1^T\mathbf{z}_1} \\ \frac{\mathbf{z}_2^T\mathbf{Y}}{\mathbf{z}_2^T\mathbf{z}_2} \\ \vdots \\ \frac{\mathbf{z}_p^T\mathbf{Y}}{\mathbf{z}_p^T\mathbf{z}_p} \end{vmatrix} = \begin{vmatrix} \frac{\mathbf{z}_1^T\mathbf{Y}}{\lambda_1} \\ \frac{\mathbf{z}_2^T\mathbf{Y}}{\lambda_2} \\ \vdots \\ \frac{\mathbf{z}_p^T\mathbf{Y}}{\lambda_p} \end{vmatrix}$$

We see that $\hat{\gamma}_i = \dfrac{\mathbf{z}_i^T\mathbf{Y}}{\mathbf{z}_i^T\mathbf{z}_i}$. So $\hat{\gamma}_i$ depends only on $\mathbf{z}_i$ and not on any other columns of $\mathbf{Z}$.

Let $\mathbf{Z}_k$ be the $n \times k$ matrix consisting of the first k columns of $\mathbf{Z}$. Let $\boldsymbol{\gamma}_k$ be the first k components of $\boldsymbol{\gamma}$. If we fit the model

$$\mathbf{Y} = \mathbf{Z}_k\boldsymbol{\gamma}_k + \boldsymbol{\varepsilon}$$

The least-squares estimate $\hat{\boldsymbol{\gamma}}_k$ will be the same as the first k components of $\hat{\boldsymbol{\gamma}}$ from the fit of the full model.

What are we doing when we drop the $\mathbf{z}_i$ with the smallest eigenvalues? First, because the components of each $\mathbf{x}_i$ average to 0, the same is true for $\mathbf{z}_i$, because $\mathbf{1}^T\mathbf{z}_i = \mathbf{1}^T\mathbf{X}\mathbf{e}_i = \mathbf{0}^T\mathbf{e}_i = 0$. Furthermore, we know that $\mathbf{z}_i^T\mathbf{z}_i = \lambda_i$. Therefore the $\mathbf{z}_i$ corresponding to small values of λ_i are the ones whose lengths are shortest, and thus those whose components tend to be small. However, we are not taking the magnitude of γ_i into account. Even though the components of $\mathbf{z}_i$ may be small, the components of $\mathbf{z}_i\gamma_i$ may be large, if γ_i is large enough.

Principal Components Regression for General X

So far we have considered only the special case where the columns of the design matrix $\mathbf{X}$

have mean 0 and variance 1. In the general case we must standardize the columns of $\mathbf{X}$. Also, the calculation of $\hat{\beta}$ from $\hat{\gamma}$ is slightly more involved.

Let $\mathbf{Y} = \beta_0\mathbf{1} + \beta_1\mathbf{x}_1 + \cdots + \beta_p\mathbf{x}_p + \boldsymbol{\varepsilon}$ be the model. Let $\bar{\mathbf{x}}_i$ be the average of the components of $\mathbf{x}_i$ and let s_i be the standard deviation. For each i, let $\mathbf{w}_i = \dfrac{\mathbf{x}_i - \bar{\mathbf{x}}_i\mathbf{1}}{s_i}$. Then $\mathbf{w}_i$ is a vector with mean 0 and standard deviation 1.

We can rewrite the model as follows:

$$\mathbf{Y} = \left(\beta_0 + \sum_{i=1}^{p} \beta_i\bar{\mathbf{x}}_i\right)\mathbf{1} + \beta_1 s_1\mathbf{w}_1 + \cdots + \beta_p s_p\mathbf{w}_p + \boldsymbol{\varepsilon}$$

Now for $i = 1, ..., p$ let $\beta_i^* = \beta_i s_i$ and let $\beta_0^* = \beta_0 + \sum_{i=1}^{p} \beta_i\bar{\mathbf{x}}_i$ The model is now

$$\mathbf{Y} = \beta_0^*\mathbf{1} + \beta_1^*\mathbf{w}_1 + \cdots + \beta_p^*\mathbf{w}_p + \boldsymbol{\varepsilon}$$

which fits the special case previously described.

Let $\mathbf{W}$ be the matrix whose columns are $\mathbf{w}_i$ and let $\boldsymbol{\Gamma}$ be the eigenvectors of $\mathbf{W}^T\mathbf{W}$. We compute the matrix $\mathbf{Z}$ as $\mathbf{Z} = \mathbf{W}\boldsymbol{\Gamma}$ and then compute the estimate $\hat{\boldsymbol{\gamma}}$ as usual, by fitting the model $\mathbf{Y} = \mathbf{Z}\boldsymbol{\gamma} + \boldsymbol{\varepsilon}$. We then compute the estimate $\hat{\beta}^* = \boldsymbol{\Gamma}\hat{\boldsymbol{\gamma}}$. Now for $i = 1, ..., p$, we can compute $\hat{\beta}_i = \hat{\beta}_i^*/s_i$. This produces the estimate $\hat{\boldsymbol{\beta}}$. Finally, we can compute $\hat{\beta}_0$ as follows: We know that $\hat{\beta}_0^* = \bar{Y}$. So we set $\hat{\beta}_0 + \sum_{i=1}^{p} \hat{\beta}_i\bar{\mathbf{x}}_i = \bar{Y}$, and obtain $\hat{\beta}_0 = \bar{Y} - \sum_{i=1}^{p} \hat{\beta}_i\bar{\mathbf{x}}_i$.

Commands in R are as follows:

```
> xm = apply(x, 2, function(y) y - mean(y))  #center the columns of X
> std = apply(x, 2, sd)                      #SDs of columns of X
> s = diag(1/std)
> w = xm%*%s
> n = nrow(w)
> gamma = eigen((n-1)*cor(w))$vectors
> z = w%*%gamma
> ghat = lm(y~z)$coef
> betastar = gamma[,1:k]%*%ghat[2:k+1] #Use only first k principal components,
                                       #omit intercept
> betahat = s%*%betastar
> oneovern = array(1/n, c(1,n))        #row vector whose components are all 1/n
> betahat0 = mean(y) - oneovern%*%x%*%betahat
```

6.5 Using Cross-Validation to Choose the Number of Principal Components

Cross-validation is often used to choose the number of principal components. Here is the step-by-step procedure for k-fold cross-validation. Let p be the number of columns of $\mathbf{X}$, so there are a maximum of p principal components.

1. Randomly split the data into k folds. Let $y_{j1}, ..., y_{jm}$ be the observed values in the jth fold.
2. For $j = 1, ..., k$:
   - For $m = 1, ..., p$:
     - Hold out fold j and perform PCR using the first m principal components on all the data except that in fold j.
     - Use the chosen model to predict the values in fold j.
     - Compute the mean squared prediction error for fold j: $\text{MSEP}_j = \frac{\sum_{i=1}^{m}(\hat{y}_{ji} - y_{ji})^2}{m}$.
   - Compute $\text{MSEP(m)} = \frac{\sum_{j=1}^{k} \text{MSEP}_j}{k}$.

The `pcr` command in R found in the `pls` library can perform the steps above, using 10-fold cross-validation. We illustrate this using the data from Example 7.1.

We begin by setting the random seed (`set.seed(1)`) so the results will be reproducible.

```
> set.seed(1)
> library(pls)
> pcr.fit = pcr(y~., data=mat, scale=TRUE, validation="CV")
> summary(pcr.fit)

Data:         X dimension: 20 10
        Y dimension: 20 1
Fit method: svdpc
Number of components considered: 10

VALIDATION: RMSEP
Cross-validated using 10 random segments.
       (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
CV           10.44    7.585    7.351    6.333    4.810    4.808    5.132
```

```
adjCV        10.44    7.454    7.277    6.209    4.765    4.785    5.050
       7 comps  8 comps  9 comps  10 comps
CV       6.006    6.458    5.794     6.697
adjCV    5.882    6.315    5.636     6.481

TRAINING: % variance explained
   1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
X    38.79    68.39    82.93    94.45    97.12    98.75    99.51    99.77
y    55.23    62.92    77.81    84.03    84.51    86.94    87.87    88.87
   9 comps  10 comps
X    99.96    100.00
y    91.12     91.75
```

The output consists of two tables. The first one presents the estimated square root of MSEP in the row labeled `CV` and another estimate that is slightly adjusted. The second table presents the percent of variance explained by each number of principal components in the row labeled `X`. The row labeled `Y` presents the R^2 of the model with each number of principal components.

Using the unadjusted estimate (CV), we see that MSEP is minimized for the model with 5 principal components.

6.6 Comparing Principal Components Regression to Linear Regression and Ridge Regression

Let $\mathbf{X}$ be the $n \times p$ centered design matrix with singular value decomposition $\mathbf{X} = \mathbf{U}\mathbf{L}\boldsymbol{\Gamma}^T$. Let $\mathbf{u}_1, ..., \mathbf{u}_p$ be the columns of $\mathbf{U}$. We have seen that the ordinary linear regression estimate of $\mathbf{X}\boldsymbol{\beta}$ is

$$\mathbf{X}\hat{\boldsymbol{\beta}}_{LR} = \mathbf{U}\mathbf{U}^T\mathbf{Y} = \sum_{i=1}^{p} \mathbf{u}_i(\mathbf{u}_i^T\mathbf{Y})$$

so $\mathbf{X}\hat{\boldsymbol{\beta}}_{LR}$ can be written as a linear combinations of the columns of $\mathbf{U}$, with the coefficient of $\mathbf{u}_i$ being $\mathbf{u}_i^T\mathbf{Y}$.

Now let $d_1 \geq d_2 \geq \cdots \geq d_p$ be the singular values of $\mathbf{X}$, so $d_1^2 \geq d_2^2 \geq \cdots \geq d_p^2$ are the eigenvalues of $\mathbf{X}^T\mathbf{X}$. Let $\mathbf{D}_\lambda$ be the diagonal matrix whose ith diagonal element is $\dfrac{d_i^2}{d_i^2 + \lambda}$ where $\lambda > 0$ We have seen that the ridge regression estimate with tuning parameter λ is

$$\mathbf{X}\hat{\boldsymbol{\beta}}_{RR} = \mathbf{U}\mathbf{D}_\lambda\mathbf{U}^T\mathbf{Y} = \sum_{i=1}^{p} \mathbf{u}_i \left(\frac{d_i^2}{d_i^2 + \lambda}\mathbf{u}_i^T\mathbf{Y}\right)$$

so the coefficients of $\mathbf{u}_i$ are shrunk by a factor $\frac{d_i^2}{d_i^2+\lambda}$. Note that the smaller d_i^2, the greater the shrinkage.

We can write the PCR estimate using k principal components as

$$\mathbf{X}\hat{\boldsymbol{\beta}}_{PCR} = \mathbf{Z}_k(\mathbf{Z}_k^T\mathbf{Z}_k)^{-1}\mathbf{Z}_k^T\mathbf{Y}$$

where $\mathbf{Z}_k$ is the matrix consisting of the first k columns of the matrix $\mathbf{Z} = \mathbf{X}\boldsymbol{\Gamma} = \mathbf{U}\mathbf{L}\boldsymbol{\Gamma}^T\boldsymbol{\Gamma} = \mathbf{U}\mathbf{L}$, where $\mathbf{L}$ is the diagonal matrix whose ith diagonal element is d_i. Therefore $\mathbf{Z}_k = \mathbf{U}_k\mathbf{L}_k$ where $\mathbf{U}_k$ is the matrix consisting of the first k columns of $\mathbf{U}$ and L_k is the upper left $k \times k$ submatrix of $\mathbf{L}$. Note that $\mathbf{U}_k^T\mathbf{U}_k = \mathbf{I}$, and $\mathbf{L}_k$ is symmetric, so $\mathbf{L}_k^T = \mathbf{L}_k$. Therefore

$$\begin{aligned}
\mathbf{X}\hat{\boldsymbol{\beta}}_{PCR} &= \mathbf{Z}_k(\mathbf{Z}_k^T\mathbf{Z}_k)^{-1}\mathbf{Z}_k^T\mathbf{Y} \\
&= \mathbf{U}_k\mathbf{L}_K(\mathbf{L}_k\mathbf{U}_k^T\mathbf{U}_k\mathbf{L}_k)^{-1}\mathbf{L}_k\mathbf{U}_k^T\mathbf{Y} \\
&= \mathbf{U}_k\mathbf{L}_K\mathbf{L}_k^{-1}\mathbf{I}\mathbf{L}_k^{-1}\mathbf{L}_k\mathbf{U}_k^T\mathbf{Y} \\
&= \mathbf{U}_k\mathbf{U}_k^T\mathbf{Y} \\
&= \sum_{i=1}^{k}\mathbf{u}_i(\mathbf{u}_i^T\mathbf{Y})
\end{aligned}$$

We see that $\mathbf{X}\hat{\boldsymbol{\beta}}_{PCR}$ can be written as a linear combinations of the first k columns of $\mathbf{U}$, with the coefficient of $\mathbf{u}_i$ being $\mathbf{u}_i^T\mathbf{Y}$.

In summary, the linear regression estimate is a linear combination of the columns of $\mathbf{U}$. Ridge regression shrinks each coefficient, with the coefficients corresponding to smaller eigenvalues being shrunk more. Principal components retains the coefficients corresponding to the largest k eigenvalues, and sets the rest equal to 0.

7 Classification

7.1 Random Vectors

Many statistical models involve several random variables. For example, we might measure height, weight, age, and income for some randomly selected people. We would model these quantities with a vector of random variables. Such random variables are said to be **jointly distributed**. A vector of jointly distributed random variables is called a **random vector**.

Random vectors may be continuous or discrete. We will focus on continuous random vectors.

Mean and Covariance
We begin by defining the mean of a random vector.

Definition 7.1. Let $\mathbf{X} = (X_1, ..., X_p)^T$ be a random vector. For each i, let $\mu_i = E(X_i)$. The expectation, or mean, of $\mathbf{X}$ is $E(\mathbf{X}) = (\mu_1, ..., \mu_p)^T$.

When we are dealing with several random variables, we need a way to measure the degree to which the value of one variable is related to the value of another. The most common such measures are the **covariance** and the **correlation**. We will focus on the covariance here.

Definition 7.2. Let X and Y be jointly distributed with means μ_X, μ_Y. The **covariance** of X and Y is $\sigma_{XY} = \text{Cov}(X, Y) = E[(X - \mu_X)(Y - \mu_Y)]$.

Proposition 7.1.
1) $\text{Cov}(X, X) = V(X)$
2) $\text{Cov}(X, Y) = \text{Cov}(Y, X)$

Definition 7.3. Let $\mathbf{X} = (X_1, ..., X_p)^T$ be a random vector. The **covariance matrix** of $\mathbf{X}$ is the $p \times p$ matrix whose ij entry is $\text{Cov}(X_i, X_j)$.

We will sometimes denote the covariance matrix of $\mathbf{X}$ by $\text{Cov}(\mathbf{X})$. Other times we will use a single letter, such as $\mathbf{\Sigma}$.

Proposition 7.2. Let $\mathbf{\Sigma}$ be the covariance matrix of $\mathbf{X}$. The diagonal elements of $\mathbf{\Sigma}$ are $V(X_1), ..., V(X_n)$.

Random Samples

Definition 7.4. Random variables $X_1, ..., X_n$ are independent and identically distributed (i.i.d.) if they are independent, and all have the same distribution. A collection of i.i.d. random variables is sometimes called a **random sample**.

Definition 7.5. Let $\mathbf{X} = (X_1, ..., X_n)^T$ be a random sample. The sample mean is $\bar{X} = \dfrac{X_1 + \cdots X_n}{n}$ and the sample variance is $s^2 = \dfrac{\sum_{i=1}^n (X_i - \bar{X})^2}{n-1}$.

Theorem 7.1. Let $\mathbf{X} = (X_1, ..., X_n)^T$ be a random sample from a population with mean μ. Then $E(\bar{X}) = \mu$.

Proof:

$$E(\bar{X}) = E\left(\frac{\sum_{i=1}^n X_i}{n}\right) = \frac{1}{n} E\left(\sum_{i=1}^n X_i\right) = \frac{1}{n}\sum_{i=1}^n E(X_i) = \frac{1}{n}\sum_{i=1}^n \mu = \mu.$$

Now imagine that we sample n individuals from a population and measure two variables X and Y on each. We then have a random sample of ordered pairs $(X_1, Y_1), ..., (X_n, Y_n)$. Let $\mu_X = E(X)$, $\mu_Y = E(Y)$, $\sigma_X^2 = V(X)$, $\sigma_Y^2 = V(Y)$, and $\text{Cov}(X, Y) = \sigma_{XY}$. We can estimate the population means and variances with the sample means and variances. We estimate the population covariance with the sample covariance, defined as follows:

Definition 7.6. Let $(X_1, Y_1), ..., (X_n, Y_n)$ be a random sample of ordered pairs. The sample covariance is $s_{XY} = \dfrac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{n-1}$.

Note that s_{XX} is the sample variance.

Just as we can sample random variables from a population of numbers, and ordered pairs from a population of ordered pairs, we can sample random vectors of any length from a population of such vectors. The following definition makes this precise.

Definition 7.7. Let $\mathbf{X}_1, ..., \mathbf{X}_n$ be independent random vectors, each with the same joint distribution. Then $\mathbf{X}_1, ..., \mathbf{X}_n$ are i.i.d., and can be referred to as a random sample.

Let's say we sample n individuals, and record values of p variables on each individual. The sample vector for the ith individual is $(X_{i1}, ..., X_{ip})$. Note that this is a row vector. The entire sample can be written as a matrix as follows:

$$\mathbf{X} = \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1p} \\ X_{21} & X_{22} & \cdots & X_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ X_{n1} & X_{n2} & \cdots & X_{np} \end{bmatrix}$$

This matrix is referred to as a **data matrix**. The entry X_{ij} represents the value of variable j for observation i. The ith row, $\mathbf{X}_{i\cdot}$ contains the values of the p variables for the ith individual, and the jth column, $\mathbf{X}_{\cdot j}$ contains the values of variable j for all n individuals.

Definition 7.8. Let $X_1, ..., X_p$ be jointly distributed random variables, and let $\mathbf{X}_1, ..., \mathbf{X}_n$ be a random sample from the distribution of $(X_1, ..., X_p)$. Denote the components of $\mathbf{X}_k$ by $\mathbf{X}_k = (X_{k1}, ..., X_{kp})$. The sample covariance of X_i and X_j is

$$s_{ij} = \frac{1}{n-1} \sum_{k=1}^{n} (X_{ki} - \bar{X}_{\cdot i})(X_{kj} - \bar{X}_{\cdot j})$$

The sample covariance matrix of $X_1, ..., X_p$ is

$$\mathbf{S} = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1p} \\ s_{21} & s_{22} & \cdots & s_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ s_{p1} & s_{p2} & \cdots & s_{pp} \end{bmatrix}$$

Note that the diagonal elements $s_{11}, ..., s_{pp}$ are the sample variances of $X_1, ..., X_p$.

The Multivariate Normal Density

Definition 7.9. Let $\boldsymbol{\mu}$ be an $n \times 1$ vector, and let $\boldsymbol{\Sigma}$ be an $n \times n$ symmetric positive definite matrix. Let $\mathbf{X}$ be an $n \times 1$ random vector whose components $X_1, ..., X_n$ have joint density

$$f(x_1, ..., x_n) = [(2\pi)^n |\boldsymbol{\Sigma}|]^{-1/2} \exp[-(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})/2]$$

Then $\mathbf{X}$ is said to have the **multivariate normal** distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, denoted $N(\boldsymbol{\mu},\ \boldsymbol{\Sigma})$.

It can be shown that $E(\mathbf{X}) = \boldsymbol{\mu}$ and $\text{Cov}(\mathbf{X}) = \boldsymbol{\Sigma}$. It can also be shown that uncorrelated multivariate normal random variables are independent. Therefore if $\boldsymbol{\Sigma} = \mathbf{I}$, $\mathbf{X}_1, ..., \mathbf{X}_n$ are independent.

7.2 Introduction to Classification

We are given a collection of items, which come from two or more previously defined populations, or groups. Classification refers to methods of determining the group to which an item belongs. The classification is done on the basis of one or more variables that are measured on each item.

Here are some examples:

- Categorize specimens of chickweed into one of several species. Variables we might consider include sepal length, petal length, and pollen diameter.
- Categorize people into good and poor credit risks. Variables we might consider include income, number of credit cards, and amount of debt.

7.3 Bayes Classifiers

Bayes classifiers can be used when the populations can be represented with known probability distributions. It is then possible to construct an optimal classifier. To make these ideas rigorous, we focus for the moment on the case where there are two groups.

Let $\mathbf{X} = (X_1, ..., X_n)$ be a vector of variables that will be measured. The possible values of $\mathbf{X}$ are divided into two disjoint regions R_1 and R_2. If $\mathbf{X} \in R_1$ the item is classified as coming from population 1, and if $\mathbf{X} \in R_2$ the item is classified as coming from population 2. The probability distribution of $\mathbf{X}$ differs between the two populations. Let $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$ be the probability density functions for populations 1 and 2, respectively. We assume that $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$ are known.

In general, no matter how R_1 and R_2 are chosen, there will be a positive probability of misclassification error. Here is an example where the probability of error is small.

Example 7.1. Values of X in population 1 are distributed $N(1, 1)$ and in population 2 are distributed $N(10, 1)$. We observe values of 2.1, 10.5, 1.1, and 9.6. How would you classify each of these items?

Solution: Values of 2.1 and 1.1 are reasonably near the center of the $N(1, 1)$ distribution but extremely far in the tail of the $N(10, 1)$ distribution. So we would classify these as coming from population 1. Similarly, we would classify 10.5 and 9.6 as coming from population 2.

In many cases the classification is not so obvious. For example, imagine that the distributions are $N(1,1)$ and $N(2,1)$. Any value between 1 and 2 could plausibly come from either distribution. No matter what rule we use for classification, there will be a substantial probability of misclassification error.

There are three factors to consider when choosing a classification rule.

1. The population pdfs $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$: Other things being equal, we would tend to classify an item in the population with the larger value for its pdf.

2. Prior probabilities for the populations: If we know that there are many more items in population 1 than in population 2 we will be more likely to classify items in population 1.

3. Costs of misclassification: There are two possible misclassification errors. We can classify an item from population 1 as coming from population 2, or we can classify an item from population 2 as coming from population 1. If one of these errors is more costly than the other, we will adjust our classification rule accordingly. For example, imagine that we are going to classify people as having a serious disease or not having it. The cost of misclassifying a sick person as healthy is greater than the cost of misclassifying a healthy person as sick. So we would tend to be more likely to classify people as diseased.

We now describe how to construct the Bayes classifier.

Let $\pi_1, ..., \pi_g$ be groups. We observe a value $\mathbf{x}$ that belongs to one of these groups and need to determine to which group it belongs. We will assume that we know the prior probabilities of the groups, the probability distribution of the values in each group, and the misclassification costs of assigning an item to the wrong group. We develop some notation.

For $i = 1, ..., g$, let p_j be the proportion of the entire population that belongs to group π_j. p_j is the prior probability of group π_j. It is the probability that a randomly sampled member of the population will belong to group π_j. Within each group, the value $\mathbf{x}$ has a probability distribution. Let $f_j(\mathbf{x})$ be the probability density function or probability mass function (usually the density) of the values in group π_j. Finally, for any two groups π_j, π_k, let $c(k|j)$ be the cost of misclassifying an item in group π_k when it is actually in group π_j.

Let $\mathbf{X}$ be a value sampled from the population. For each group π_j, the probability that $\mathbf{X}$ is in group π_j is $p_j = P(\mathbf{X} \in \pi_j)$. Let $\mathbf{x}$ be the observed value of $\mathbf{X}$. Now the probability that $\mathbf{X}$ is in group π_j is the conditional probability $P(\mathbf{X} \in \pi_j | \mathbf{X} = \mathbf{x})$. Assume for the moment that $\mathbf{X}$ is discrete. We use Bayes' rule to compute the conditional probability that $\mathbf{X} \in \pi_j$.

$$P(\mathbf{X} \in \pi_j | \mathbf{X} = \mathbf{x}) = \frac{P(\mathbf{X} = \mathbf{x} | \mathbf{X} \in \pi_j) p_j}{\sum_{i=1}^{g} P(\mathbf{X} = \mathbf{x} | \mathbf{X} \in \pi_i) p_i}$$

When $\mathbf{X}$ is continuous, which is usually the case, we replace $P(\mathbf{X} = \mathbf{x} \,|\, \mathbf{X} \in \pi_j)$ with $f_j(\mathbf{x})$ to obtain

$$P(\mathbf{X} \in \pi_j | \mathbf{X} = \mathbf{x}) = \frac{f_j(\mathbf{x}) p_j}{\sum_{i=1}^{g} f_i(\mathbf{x}) p_i}$$

If we classify $\mathbf{x}$ in group k, the expected cost of misclassification (ECM) is

$$ECM_k = \sum_{j \neq k} c(k|j) P(\mathbf{X} \in \pi_j | \mathbf{X} = \mathbf{x}) = \sum_{j \neq k} \frac{c(k|j) f_j(\mathbf{x}) p_j}{\sum_{i=1}^{g} f_i(\mathbf{x}) p_i}$$

The optimal classifier assigns $\mathbf{x}$ to the class k for which ECM_k is a minimum. Note that the denominator $\sum_{i=1}^{g} f_i(\mathbf{x}) p_i$ does not depend on k, so the optimal rule assigns $\mathbf{x}$ to the class k for which

$$\sum_{j \neq k} c(k|j) f_j(\mathbf{x}) p_j$$

is a minimum.

This classification method is called the *Bayes classifier*. When there are only two classes the rule reduces to the following:

$$\text{Assign } \mathbf{x} \text{ to class 1 if } c(2|1) p_1 f_1(\mathbf{x}) > \; c(1|2) p_2 f_2(\mathbf{x})$$

otherwise assign $\mathbf{x}$ to class 2.

The Bayes Classifier When Misclassification Costs are Equal

When the costs $c(k|j)$ are the same for all $j \neq k$, the Bayes classifier is the classifier that minimizes the probability of incorrect classification, also called the *expected misclassification rate*.

Theorem 7.2. Assume we observe $\mathbf{X}$ from a population consisting of groups $\pi_1, ..., \pi_g$. For each j, let p_j be the prior probability of group j and let $f_j(\mathbf{x})$ be the density of $\mathbf{x}$ in group j.

Assume the misclassification costs $c(k|j)$ are equal for all $j \neq k$. Then the Bayes classifier minimizes the expected misclassification rate.

Proof:
The Bayes classifier minimizes $\sum_{j \neq k} f_j(\mathbf{x}) p_j c(k|j)$. Since $c(k|j)$ are equal for all $j \neq k$, the Bayes classifier minimizes $\sum_{j \neq k} f_j(\mathbf{x}) p_j$.

The expected misclassification rate when $\mathbf{x}$ is classified in group k is $P(\mathbf{X} \notin \pi_k | \mathbf{X} = \mathbf{x}) = \sum_{j \neq k} P(\mathbf{X} \in \pi_j | \mathbf{X} = \mathbf{x}) = \dfrac{f_j(\mathbf{x}) p_j}{\sum_{i=1}^{g} f_i(\mathbf{x}) p_i}$. Since the Bayes classifier minimizes the numerator of this expression, and the denominator does not depend on k, the Bayes estimator minimizes the expected misclassification rate.

When misclassification costs are equal the Bayes classifier can be expressed in a simple form. We have seen in the proof of Theorem 7.2 that the Bayes classifier assigns $\mathbf{x}$ to the group k for which $\sum_{j \neq k} f_j(\mathbf{x}) p_j$ is a minimum, Now

$$\sum_{j \neq k} f_j(\mathbf{x}) p_j = \sum_{i=1}^{g} f_i(\mathbf{x}) p_i - f_k(\mathbf{x}) p_k$$

Since $\sum_{i=1}^{g} f_i(\mathbf{x}) p_i$ does not depend on k, this is equivalent to maximizing $f_k(\mathbf{x}) p_k$. Therefore we can define the Bayes classifier as follows:

When misclassification costs are equal, the Bayes classifier assigns $\mathbf{x}$ to the group k for which $f_k(\mathbf{x}) p_k$ is a maximum.

Note that the posterior probability that $\mathbf{X}$ belongs to group k is $P(\mathbf{X} \in \pi_k | \mathbf{X} = \mathbf{x}) = \dfrac{f_k(\mathbf{x}) p_k}{\sum_{i=1}^{g} f_i(\mathbf{x}) p_i}$. Since $\sum_{i=1}^{g} f_i(\mathbf{x}) p_i$ does not depend on k, we see that the Bayes classifier maximizes the posterior probability. We conclude

When misclassification costs are equal, the Bayes classifier assigns $\mathbf{x}$ to the group k with largest posterior probability.

Example 7.2. Let π_1, ..., π_g be multivariate normal populations with common covariance matrix $\mathbf{\Sigma}$. Let $p_1, ..., p_g$ be the prior probabilities and assume the misclassification costs are equal. Find the Bayes classifier.

Solution:
Let $\boldsymbol{\mu}_1$, ..., $\boldsymbol{\mu}_p$ be the population means, let $\mathbf{x}$ be an observation, and let p be the dimension of $\mathbf{x}$. We want to maximize $f_k(\mathbf{x})p_k$, or equivalently $\log f_k(\mathbf{x}) + \log p_k$. Now

$$\log f_k(\mathbf{x}) = \frac{p}{2}\log 2\pi - \frac{1}{2}\log|\boldsymbol{\Sigma}| - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)$$

The Bayes classifier maximizes $-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)+\log p_k$, or equivalently, minimizes $\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_k) - \log p_k$.

Computing the Bayes classifier requires that prior probabilities, densities within each group, and misclassification costs are known. In practice it is often the case that none of these are known exactly. In particular, the densities within each group are usually unknown, so they are estimated from the information in a training set. One of the most commonly used Bayes classifiers is linear discriminant analysis.

7.4 Linear Discriminant Analysis

Linear discriminant analysis (LDA) refers to the Bayes classifier that results when the groups are all assumed to be multivariate normal with the same covariance matrix. Misclassification costs are also assumed to be equal.

Let μ_i be the mean of the the density f_i and let $\boldsymbol{\Sigma}$ be the common covariance matrix. The densities are given by

$$f_i(\mathbf{x}) = \frac{1}{(2\pi)^{p/2}|\boldsymbol{\Sigma}|^{1/2}}\exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_i)\right]$$

Let $p_1, ..., p_g$ be the prior probabilities. The Bayes classifier assigns $\mathbf{x}$ to the group π_k that maximizes $f_k(\mathbf{x})p_k$. Equivalently the Bayes classifier maximizes

$$\log f_k(\mathbf{x}) + \log p_k = \frac{p}{2}\log 2\pi - \frac{1}{2}\log|\boldsymbol{\Sigma}| - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_k) + \log p_k$$

Now $\frac{p}{2}\log 2\pi - \frac{1}{2}\log|\boldsymbol{\Sigma}|$ does not depend on k. Multiplying the remaining terms by -1, we see that the Bayes classifier *minimizes*

$$\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_k) - \log p_k$$

In practice, we do not know, $\boldsymbol{\mu}_1$, ..., $\boldsymbol{\mu}_g$, or $\boldsymbol{\Sigma}$. So we must estimate them from a training set. Assume we have a random sample of n_i items from π_i for $i = 1, ..., p$.

For each i, Let $\mathbf{x}_{i1}, \mathbf{x}_{i2}, ..., \mathbf{x}_{in_i}$ be the sample from π_i. Let

$$\mathbf{X}_i = \begin{bmatrix} \mathbf{x}_{i1}^{'} \\ \mathbf{x}_{i2}^{'} \\ \vdots \\ \mathbf{x}_{in_i}^{'} \end{bmatrix}$$

be the data matrix for the sample from group π_i.

Each row of $\mathbf{X}_i$ is a vector sampled from π_i. It follows that the average of the jth column is the sample mean of the jth components of the sample vectors. The sample mean vector for group π_i is therefore

$$\bar{\mathbf{x}}_i = \frac{1}{n_i} \sum_{j=1}^{n_i} \mathbf{x}_{ij}$$

$\bar{\mathbf{x}}_i$ is a $p \times 1$ vector of means. Let $\mathbf{S}_i$ be the sample covariance matrix for $\mathbf{X}_i$.

Since we assume that all the groups have the same covariance, we must combine $\mathbf{S}_1$, ..., $\mathbf{S}_g$ to obtain a single estimate of the common covariance. The result is the **pooled** estimate

$$\mathbf{S}_{pooled} = \frac{(n_1 - 1)\mathbf{S}_1 + \cdots + (n_g - 1)\mathbf{S}_g}{n_1 + \cdots n_g - g}$$

The (LDA) classifier assigns $\mathbf{x}$ to the group π_k for which

$$\frac{1}{2}(\mathbf{x} - \bar{\mathbf{x}}_k)^T \mathbf{S}_{pooled}^{-1} (\mathbf{x} - \bar{\mathbf{x}}_k) - \log p_k$$

is a minimum.

Is the LDA classifier really linear?
At first the LDA classifier does not look linear, because $(\mathbf{x} - \bar{\mathbf{x}}_k)^T \mathbf{S}_{pooled}^{-1} (\mathbf{x} - \bar{\mathbf{x}}_k)$ is a quadratic form. However, it is linear. To see this, multiply it out.

$$\begin{aligned} (\mathbf{x} - \bar{\mathbf{x}}_k)^T \mathbf{S}_{pooled}^{-1} (\mathbf{x} - \bar{\mathbf{x}}_k) &= \mathbf{x}^T \mathbf{S}_{pooled}^{-1} \mathbf{x} - \bar{\mathbf{x}}_k^T \mathbf{S}_{pooled}^{-1} \mathbf{x} - \mathbf{x}^T \mathbf{S}_{pooled}^{-1} \bar{\mathbf{x}}_k + \bar{\mathbf{x}}_k^T \mathbf{S}_{pooled}^{-1} \bar{\mathbf{x}}_k \\ &= \mathbf{x}^T \mathbf{S}_{pooled}^{-1} \mathbf{x} - 2\bar{\mathbf{x}}_k^T \mathbf{S}_{pooled}^{-1} \mathbf{x} + \bar{\mathbf{x}}_k^T \mathbf{S}_{pooled}^{-1} \bar{\mathbf{x}}_k \end{aligned}$$

The first term is quadratic in $\mathbf{x}$, but it does not depend on k. Therefore, the LDA classifier assigns $\mathbf{x}$ to the group for which

$$\frac{1}{2}(-2\bar{\mathbf{x}}_k^T\mathbf{S}_{pooled}^{-1}\mathbf{x} + \bar{\mathbf{x}}_k^T\mathbf{S}_{pooled}^{-1}\bar{\mathbf{x}}_k) - \log p_k$$

is a minimum. This quantity is linear in $\mathbf{x}$. Thus the classifier is a linear function of the vector to be classified.

Example 7.3. To detect potential hemophilia carriers, blood samples were assayed for two samples of women. Group 1 is a group of non-carriers, and group 2 is a group of carriers. Two variables measured the anti-hemophiliac factor (AHF) on each woman in each group. They were

$$x_1 = \log_{10}(\text{AHF activity})$$
$$x_2 = \log_{10}(\text{AHF antigen})$$

The sample sizes were $n_1 = 30$ and $n_2 = 22$. The sample means are:

$$\bar{\mathbf{x}}_1 = \begin{bmatrix} -0.0065 \\ -0.0390 \end{bmatrix} \qquad \bar{\mathbf{x}}_2 = \begin{bmatrix} -0.2483 \\ 0.0262 \end{bmatrix}$$

The inverse of the pooled covariance matrix is

$$\mathbf{S}_{pooled}^{-1} = \begin{bmatrix} 131.158 & -90.423 \\ -90.423 & 108.147 \end{bmatrix}$$

Find the estimated optimal classification rule, assuming equal prior probabilities and equal misclassification costs.

Solution:

Let $\mathbf{x}_0 = [x_1\ x_2]^T$ be the item to be classified. We compute the values of the classifier for both groups. Since the priors are equal, we can omit the term $\log p_k$. We can also ignore the factor $1/2$.

$$\text{For group 1: } -2\bar{\mathbf{x}}_1^T\mathbf{S}_{pooled}^{-1}\mathbf{x} + \bar{\mathbf{x}}_1^T\mathbf{S}_{pooled}^{-1}\bar{\mathbf{x}}_1 = -5.34794x_1 + 7.259967x_2 + 0.1241886$$

$$\text{For group 2: } -2\bar{\mathbf{x}}_2^T\mathbf{S}_{pooled}^{-1}\mathbf{x} + \bar{\mathbf{x}}_2^T\mathbf{S}_{pooled}^{-1}\bar{\mathbf{x}}_2 = 69.87123x_1 - 50.57096x_2 + 9.336993$$

We assign $\mathbf{x}_0$ to group 1 if
$-5.34794x_1 + 7.259967x_2 + 0.1241886 < 69.87123x_1 - 50.57096x_2 + 9.336993$.
Otherwise we assign $\mathbf{x}_0$ to group 2.

Example 7.4. For a particular woman, the value of $\mathbf{x}_0$ is $\mathbf{x}_0 = [-0.210, -0.044]^T$. Assuming equal costs and equal priors, how should this woman be classified?

Solution:
We substitute $x_1 = -0.210$ and $x_2 = -0.044$.
$-5.34794(-0.210) + 7.259967(-0.044) + 0.1241886 = 0.9278175$.
$69.87123(-0.210) - 50.57096(-0.044) + 9.336993 = -3.110843$.
The woman is classified as a carrier.

Example 7.5. Assume that in fact, that only 1 in 2500 women is a carrier. Find the estimated optimal allocation rule, assuming equal misclassification costs. How should a woman with $\mathbf{x}_0 = [-0.210, -0.044]^T$ be classified?

Solution:
The prior probabilities are $p_1 = 2499/2500$, $p_2 = 1/2500$. The estimated optimal allocation rule is to assign to group 1 if

$$-5.34794x_1+7.259967x_2+0.1241886-\log(2499/2500) < 69.87123x_1-50.57096x_2+9.336993-\log(1/2500)$$

Otherwise we assign it to group 2. Substituting $x_1 = -0.210$, $x_2 = -0.044$, we have
$-5.34794(-0.210) + 7.259967(-0.044) + 0.1241886 - \log(2499/2500) = 0.9282175$
and $69.87123(-0.210) - 50.57096(-0.044) + 9.336993 - \log(1/2500) = 4.713203$.
This woman is classified as a non-carrier.

Example 7.6. Following are data on income and lot size for a random sample of 12 owners of riding mowers and for 12 non-owners. Formulate the estimated optimal classification rule based on the assumption that the populations are both bivariate normal with equal covariance matrices.

Riding Mower Owners (π_1)		Non-Owners (π_2)	
X_1 (Income in \$1000s)	X_2 (Lot Size in 1000 ft^2)	X_1 (Income in \$1000s)	X_2 (Lot Size in 1000 ft^2)
90.0	18.4	105.0	19.6
115.5	16.8	82.8	20.8
94.8	21.6	94.8	17.2
91.5	20.8	73.2	20.4
117.0	23.6	114.0	17.6
140.1	19.2	79.2	17.6
138.0	17.6	89.4	16.0
112.8	22.4	96.0	18.4
99.0	20.0	77.4	16.4
123.0	20.8	63.0	18.8
81.0	22.0	81.0	14.0
111.0	20.0	93.0	14.8

Solution:

We compute the sample means and covariances:

$$\bar{\mathbf{x}}_1 = (109.475, 20.467)^T \qquad \bar{\mathbf{x}}_2 = (87.400, 17.633)^T$$

$$\mathbf{S}_1 = \begin{bmatrix} 352.644 & -11.818 \\ -11.818 & 4.082 \end{bmatrix} \qquad \mathbf{S}_2 = \begin{bmatrix} 200.705 & -2.589 \\ -2.589 & 4.464 \end{bmatrix}$$

$$\mathbf{S}_{pooled} = \frac{12-1}{24-2}\mathbf{S}_1 + \frac{12-1}{24-2}\mathbf{S}_2 = \begin{bmatrix} 276.675 & -7.204 \\ -7.204 & 4.273 \end{bmatrix}$$

$$\mathbf{S}_{pooled}^{-1} = \begin{bmatrix} 0.00378027 & 0.00637247 \\ 0.00637247 & 0.244752 \end{bmatrix}$$

Let $\mathbf{x}_0 = [x_1 \ x_2]^T$ be the item to be classified. We compute the values of the classifier for both groups. Since the priors are equal, we can omit the term $\log p_k$. We can also ignore the factor $1/2$.

$$-2\bar{\mathbf{x}}_1^T\mathbf{S}_{pooled}^{-1}\mathbf{x} + \bar{\mathbf{x}}_1^T\mathbf{S}_{pooled}^{-1}\bar{\mathbf{x}}_1 = -1.088541x_1 - 11.41393x_2 + 176.3885$$

$$-2\bar{\mathbf{x}}_2^T\mathbf{S}_{pooled}^{-1}\mathbf{x} + \bar{\mathbf{x}}_2^T\mathbf{S}_{pooled}^{-1}\bar{\mathbf{x}}_2 = -0.8855227x_1 - 9.745332x_2 + 124.6171$$

We assign $\mathbf{x}_0$ to group 1 if
$-1.088541x_1 - 11.41393x_2 + 176.3885 < -0.8855227x_1 - 9.745332x_2 + 124.6171$.
Otherwise we assign $\mathbf{x}_0$ to group 2.

7.5 Estimating the Misclassification Rate

When population distributions and prior distributions are known, it is straightforward to calculate the expected misclassification rate (EMR) for any classification rule. In practice, however, these distributions must be estimated, and the expected misclassification rate must be estimated rather than computed exactly.

Let the classes be π_1, ..., π_g. For each pair of classes π_i, π_j, let $P(\pi_j \mid \pi_i)$ be the probability that an item actually in class π_i is classified in π_j. The probability that an item in class i is correctly classified is $P(\pi_i \mid \pi_i)$. Now let $p_1, ..., p_k$ be the prior probabilities of the classes. The probability that a randomly chosen item is correctly classified is $\sum_{i=1}^{k} P(\pi_i \mid \pi_i)p_i$. The probability that a randomly chosen item is misclassified is $1 - \sum_{i=1}^{k} P(\pi_i \mid \pi_i)p_i$. This is the misclassification rate.

The most effective method for estimating a misclassification rate is through cross-validation. The training set is divided randomly into a number k of folds. Each fold is withheld in turn, and the items in the fold are classified using the items in the other folds. The proportion of items in the entire training set that are misclassified is used to estimate the misclassification rate. Here are the steps in the procedure.

Let n be the total number of observations in the training set. Let k be the number of folds ($k = 10$ is a popular choice).

1. Divide the training set randomly into k folds.
2. For $i = 1$ to k:
   (a) Omit the kth fold from the training set.
   (b) Use the observations in the remaining $k - 1$ folds to develop a classification rule.
   (c) Use the rule to classify the items in the omitted fold.
3. For each class π_i, and each $j \neq i$, let n_{ij} be the number of items in π_i that were classified in π_j. The misclassification rate is estimated as $\dfrac{\sum_{i=1}^{n} \sum_{j \neq i} n_{ij}}{n}$.

A popular choice for the number of folds is $k = 10$. Another popular method is leave-one-out cross-validation (LOOCV), in which each item is in its own fold.

Example 7.7. Use LOOCV to estimate the misclassification rates $P(\pi_2 \mid \pi_1)$ and $P(\pi_1 \mid \pi_2)$ for the mower data.

Solution:
We use R. Here are the data in a file called "mower.dat."

Category	Income	Lot Size	Category	Income	Lot Size
1	90.0	18.4	2	105.0	19.6
1	115.5	16.8	2	82.8	20.8
1	94.8	21.6	2	94.8	17.2
1	91.5	20.8	2	73.2	20.4
1	117.0	23.6	2	114.0	17.6
1	140.1	19.2	2	79.2	17.6
1	138.0	17.6	2	89.4	16.0
1	112.8	22.4	2	96.0	18.4
1	99.0	20.0	2	77.4	16.4
1	123.0	20.8	2	63.0	18.8
1	81.0	22.0	2	81.0	14.0
1	111.0	20.0	2	93.0	14.8

```
> data = read.csv("mower.csv")
> n = nrow(mower)

> pcv = rep(0, n)  #Initialize array of predicted classes

> for (i in 1:n) {
+ cvdata = mower[-i,]  #Remove the ith item

#Develop classification scheme based on all but ith item.
+ zcv = lda(mower[-i, "Category"]~., mower[-i,2:3], prior=c(0.5, 0.5))

+ ppcv = predict(zcv, mower[i, 2:3])  #Classify ith item
+ pcv[i] = ppcv$class
+ }

> table(mower$Category, pcv)  #Construct confusion matrix
   pcv
     1  2
  1 10  2
  2  3  9

#The misclassification rate is estimated to be 5/24 = 0.20833.
```

Example 7.8. Fisher's iris data set contains measurements on sepal length, sepal width, petal length, and petal width for 150 irises. There are 50 plants in each of three species. The data are presented below.

Sepal Length	Sepal Width	Petal Length	Petal Width	Species	Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	setosa	6.6	3.0	4.4	1.4	versicolor
4.9	3.0	1.4	0.2	setosa	6.8	2.8	4.8	1.4	versicolor
4.7	3.2	1.3	0.2	setosa	6.7	3.0	5.0	1.7	versicolor
4.6	3.1	1.5	0.2	setosa	6.0	2.9	4.5	1.5	versicolor
5.0	3.6	1.4	0.2	setosa	5.7	2.6	3.5	1.0	versicolor
5.4	3.9	1.7	0.4	setosa	5.5	2.4	3.8	1.1	versicolor
4.6	3.4	1.4	0.3	setosa	5.5	2.4	3.7	1.0	versicolor
5.0	3.4	1.5	0.2	setosa	5.8	2.7	3.9	1.2	versicolor
4.4	2.9	1.4	0.2	setosa	6.0	2.7	5.1	1.6	versicolor
4.9	3.1	1.5	0.1	setosa	5.4	3.0	4.5	1.5	versicolor
5.4	3.7	1.5	0.2	setosa	6.0	3.4	4.5	1.6	versicolor
4.8	3.4	1.6	0.2	setosa	6.7	3.1	4.7	1.5	versicolor
4.8	3.0	1.4	0.1	setosa	6.3	2.3	4.4	1.3	versicolor
4.3	3.0	1.1	0.1	setosa	5.6	3.0	4.1	1.3	versicolor
5.8	4.0	1.2	0.2	setosa	5.5	2.5	4.0	1.3	versicolor
5.7	4.4	1.5	0.4	setosa	5.5	2.6	4.4	1.2	versicolor
5.4	3.9	1.3	0.4	setosa	6.1	3.0	4.6	1.4	versicolor
5.1	3.5	1.4	0.3	setosa	5.8	2.6	4.0	1.2	versicolor
5.7	3.8	1.7	0.3	setosa	5.0	2.3	3.3	1.0	versicolor
5.1	3.8	1.5	0.3	setosa	5.6	2.7	4.2	1.3	versicolor
5.4	3.4	1.7	0.2	setosa	5.7	3.0	4.2	1.2	versicolor
5.1	3.7	1.5	0.4	setosa	5.7	2.9	4.2	1.3	versicolor
4.6	3.6	1.0	0.2	setosa	6.2	2.9	4.3	1.3	versicolor
5.1	3.3	1.7	0.5	setosa	5.1	2.5	3.0	1.1	versicolor
4.8	3.4	1.9	0.2	setosa	5.7	2.8	4.1	1.3	versicolor
5.0	3.0	1.6	0.2	setosa	6.3	3.3	6.0	2.5	virginica
5.0	3.4	1.6	0.4	setosa	5.8	2.7	5.1	1.9	virginica
5.2	3.5	1.5	0.2	setosa	7.1	3.0	5.9	2.1	virginica
5.2	3.4	1.4	0.2	setosa	6.3	2.9	5.6	1.8	virginica
4.7	3.2	1.6	0.2	setosa	6.5	3.0	5.8	2.2	virginica
4.8	3.1	1.6	0.2	setosa	7.6	3.0	6.6	2.1	virginica
5.4	3.4	1.5	0.4	setosa	4.9	2.5	4.5	1.7	virginica
5.2	4.1	1.5	0.1	setosa	7.3	2.9	6.3	1.8	virginica
5.5	4.2	1.4	0.2	setosa	6.7	2.5	5.8	1.8	virginica
4.9	3.1	1.5	0.2	setosa	7.2	3.6	6.1	2.5	virginica
5.0	3.2	1.2	0.2	setosa	6.5	3.2	5.1	2.0	virginica
5.5	3.5	1.3	0.2	setosa	6.4	2.7	5.3	1.9	virginica
4.9	3.6	1.4	0.1	setosa	6.8	3.0	5.5	2.1	virginica
4.4	3.0	1.3	0.2	setosa	5.7	2.5	5.0	2.0	virginica
5.1	3.4	1.5	0.2	setosa	5.8	2.8	5.1	2.4	virginica
5.0	3.5	1.3	0.3	setosa	6.4	3.2	5.3	2.3	virginica
4.5	2.3	1.3	0.3	setosa	6.5	3.0	5.5	1.8	virginica
4.4	3.2	1.3	0.2	setosa	7.7	3.8	6.7	2.2	virginica
5.0	3.5	1.6	0.6	setosa	7.7	2.6	6.9	2.3	virginica
5.1	3.8	1.9	0.4	setosa	6.0	2.2	5.0	1.5	virginica
4.8	3.0	1.4	0.3	setosa	6.9	3.2	5.7	2.3	virginica
5.1	3.8	1.6	0.2	setosa	5.6	2.8	4.9	2.0	virginica
4.6	3.2	1.4	0.2	setosa	7.7	2.8	6.7	2.0	virginica
5.3	3.7	1.5	0.2	setosa	6.3	2.7	4.9	1.8	virginica
5.0	3.3	1.4	0.2	setosa	6.7	3.3	5.7	2.1	virginica
7.0	3.2	4.7	1.4	versicolor	7.2	3.2	6.0	1.8	virginica
6.4	3.2	4.5	1.5	versicolor	6.2	2.8	4.8	1.8	virginica
6.9	3.1	4.9	1.5	versicolor	6.1	3.0	4.9	1.8	virginica
5.5	2.3	4.0	1.3	versicolor	6.4	2.8	5.6	2.1	virginica
6.5	2.8	4.6	1.5	versicolor	7.2	3.0	5.8	1.6	virginica
5.7	2.8	4.5	1.3	versicolor	7.4	2.8	6.1	1.9	virginica
6.3	3.3	4.7	1.6	versicolor	7.9	3.8	6.4	2.0	virginica
4.9	2.4	3.3	1.0	versicolor	6.4	2.8	5.6	2.2	virginica
6.6	2.9	4.6	1.3	versicolor	6.3	2.8	5.1	1.5	virginica
5.2	2.7	3.9	1.4	versicolor	6.1	2.6	5.6	1.4	virginica
5.0	2.0	3.5	1.0	versicolor	7.7	3.0	6.1	2.3	virginica
5.9	3.0	4.2	1.5	versicolor	6.3	3.4	5.6	2.4	virginica
6.0	2.2	4.0	1.0	versicolor	6.4	3.1	5.5	1.8	virginica
6.1	2.9	4.7	1.4	versicolor	6.0	3.0	4.8	1.8	virginica
5.6	2.9	3.6	1.3	versicolor	6.9	3.1	5.4	2.1	virginica
6.7	3.1	4.4	1.4	versicolor	6.7	3.1	5.6	2.4	virginica
5.6	3.0	4.5	1.5	versicolor	6.9	3.1	5.1	2.3	virginica
5.8	2.7	4.1	1.0	versicolor	5.8	2.7	5.1	1.9	virginica
6.2	2.2	4.5	1.5	versicolor	6.8	3.2	5.9	2.3	virginica
5.6	2.5	3.9	1.1	versicolor	6.7	3.3	5.7	2.5	virginica
5.9	3.2	4.8	1.8	versicolor	6.7	3.0	5.2	2.3	virginica
6.1	2.8	4.0	1.3	versicolor	6.3	2.5	5.0	1.9	virginica
6.3	2.5	4.9	1.5	versicolor	6.5	3.0	5.2	2.0	virginica
6.1	2.8	4.7	1.2	versicolor	6.2	3.4	5.4	2.3	virginica
6.4	2.9	4.3	1.3	versicolor	5.9	3.0	5.1	1.8	virginica

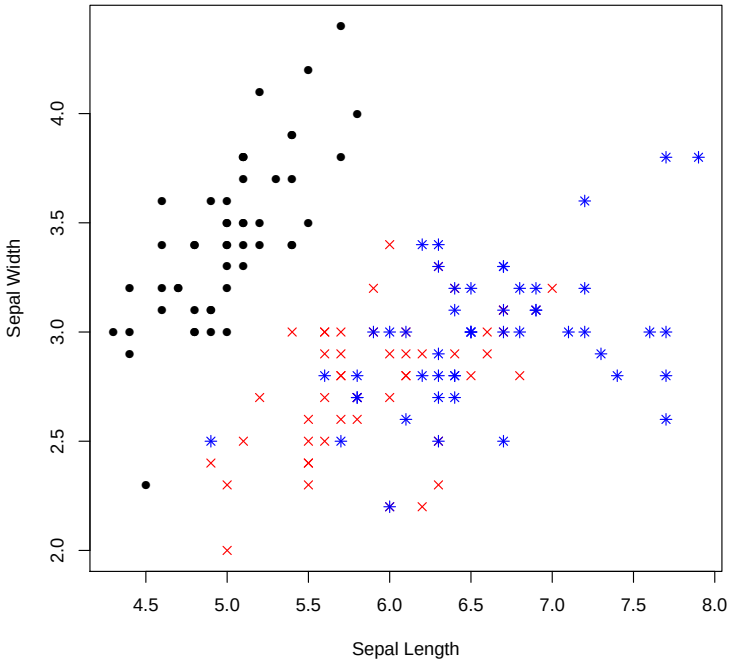
The 150 irises with known species are a training set. Here are data for 10 new irises, whose species are unknown.

Sepal Length	Sepal Width	Petal Length	Petal Width
5.9	3.5	1.6	0.2
5.2	3.4	1.5	0.2
3.9	2.7	1.7	0.2
4.5	1.9	3.2	1.1
6.5	3.7	4.3	1.4
6.0	2.2	3.6	1.8
7.5	3.0	4.3	1.5
5.6	3.2	5.6	2.0
7.4	2.8	4.4	2.2
7.0	3.4	4.6	2.0

Use linear discriminant analysis on the training set to develop a classification rule. Then use this rule to classify the new plants:
1) Using sepal length and width
2) Using petal length and width

Estimate the misclassification rate for each rule using both LOOCV and 10-fold cross-validation.

Solution:
Following is a plot of sepal width versus sepal length, with different symbols used for each species. It is clear that one species (setosa) is fairly well differentiated, but the other two overlap some.



We use R to perform LDA. The 150 items in the training set are in a file called `iris`, and the new plants to be classified are in a file called `newiris`.

```
> library(MASS)

> zcv = lda(iris$Species~., iris[,1:2], prior=c(1/3, 1/3, 1/3))
+ ppcv = predict(zcv, newiris[, 1:2])

> ppcv$class
 [1] setosa  setosa  setosa  versicolor virginica  versicolor virginica
 [8] setosa  virginica  virginica
Levels: setosa versicolor virginica

#Estimate the misclassification rate using LOOCV.

> n = length(iris$Species)
> pcv = rep(0, n)
```

```
> for (i in 1:n) {
+ zcv = lda(iris[-i, "Species"]~., iris[-i,1:2], prior=c(1/3, 1/3, 1/3))
+ ppcv = predict(zcv, iris[i, 1:2])
+ pcv[i] = ppcv$class
+ }

> table(iris$Species, pcv)
            pcv
              1  2  3
   setosa     49  1  0
   versicolor  0 35 15
   virginica   0 15 35
```

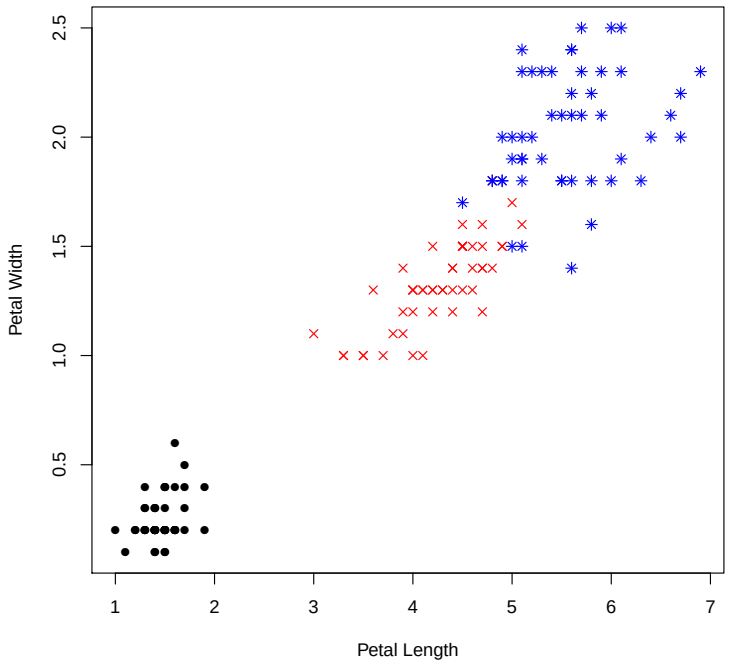
The estimated misclassification rate is $31/150 = 0.2067$.

Now we'll estimate the misclassification rate with 10-fold cross-validation.

```
> set.seed(1)
> k = 10
> n = length(iris$Species)
> fold = sample(k, n, replace=TRUE)

> confuse = matrix(rep(0, 9), nrow=3)  #initialize confusion matrix

> for (i in 1:k) {
> zcv = lda(iris[fold != i, "Species"]~., iris[fold !=i, 1:2], prior=c(1/3, 1/3, 1/3))
> ppcv = predict(zcv, iris[fold==i, 1:2])
> confuse = confuse + table(iris[fold==i, "Species"], ppcv$class)
}

> confuse
             setosa  versicolor  virginica
   setosa          49           1          0
   versicolor       0          35         15
   virginica        0          14         36
```

The estimated misclassification rate using 10-fold cross-validation is the same as that using LOOCV.

Now we'll try to classify using petal length and width. First a plot. The species are better differentiated by petals than by sepals. We will expect the misclassification rates to be lower.




```
> library(MASS)

> zcv = lda(iris$Species~., iris[,3:4], prior=c(1/3, 1/3, 1/3))
+ ppcv = predict(zcv, newiris[, 3:4])

> ppcv$class
 [1] setosa setosa setosa versicolor versicolor versicolor versicolor
 [8] virginica virginica  virginica
Levels: setosa versicolor virginica

#Estimate the misclassification rate with LOOCV.

> n = length(iris$Species)
> pcv = rep(0, n)
```

```
> for (i in 1:n) {
+ zcv = lda(iris[-i, "Species"]~., iris[-i,3:4], prior=c(1/3, 1/3, 1/3))
+ ppcv = predict(zcv, iris[i, 3:4])
+ pcv[i] = ppcv$class
+ }

> table(iris$Species, pcv)
            pcv
              1  2  3
   setosa     50  0  0
   versicolor  0 48  2
   virginica   0  4 46
```

LOOCV estimates the misclassification rate to be $6/150 = 0.04$.

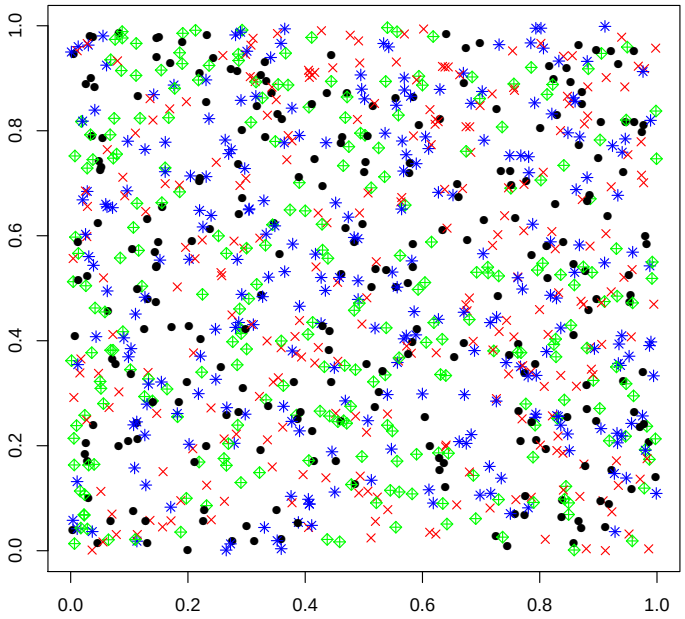
Now we'll try 10-fold cross-validation.

```
> set.seed(1)
> k = 10
> n = length(iris$Species)
> fold = sample(k, n, replace=TRUE)

> confuse = matrix(rep(0, 9), nrow=3)  #initialize confusion matrix

> for (i in 1:k) {
> zcv = lda(iris[fold != i, "Species"]~., iris[fold !=i, 1:4], prior=c(1/3, 1/3, 1/3))
> ppcv = predict(zcv, iris[fold==i, 1:4])
> confuse = confuse + table(iris[fold==i, "Species"], ppcv$class)

> confuse
              setosa  versicolor  virginica
   setosa         50           0          0
   versicolor      0          48          2
   virginica       0           1         49
}
```

10-fold cross-validation estimates the misclassification rate to be $3/150 = 0.02$.

Example 7.9. The following plot presents 1000 points uniformly distributed on the unit square. The points have been randomly divided into four groups of 250 points each, as

indicated by the different symbols on the plot. Fifty points from each group have been chosen at random to form a test set. Use LDA to classify the remaining points. Compute the misclassification rate on the test set, and use cross-validation to estimate the misclassification rate.

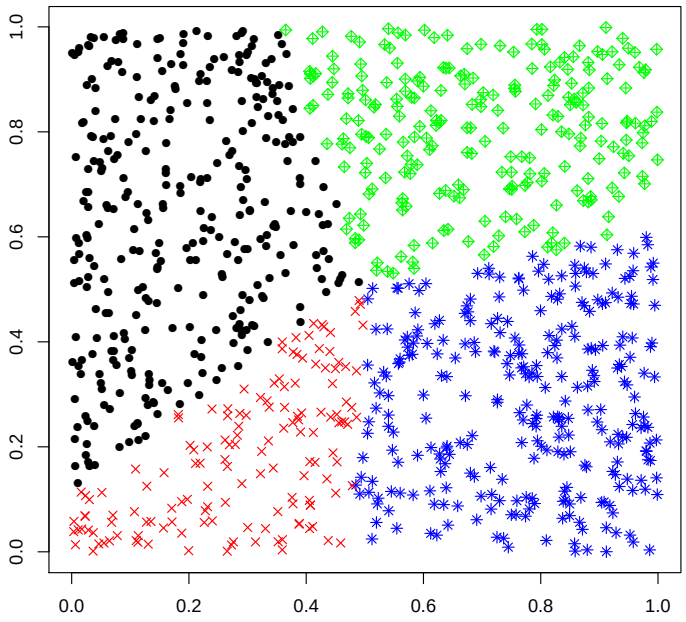


Solution: We use R:

```
#yy = 1000 uniform points on the unit square.
#yclass = true class for each point.
#train is the training set consisting of 50 points in each group.

> zrandom = lda(yclass ~ ., yy)
> prandom = predict(zrandom, yy)

> table(yclass, prandom$class)

     1  2  3  4
  1 31 81 55 75
  2 40 83 56 71
  3 34 60 70 86
  4 33 69 58 90
```

Misclassification Rate: $726/1000 = 0.726$

Here is a plot showing the classifications.



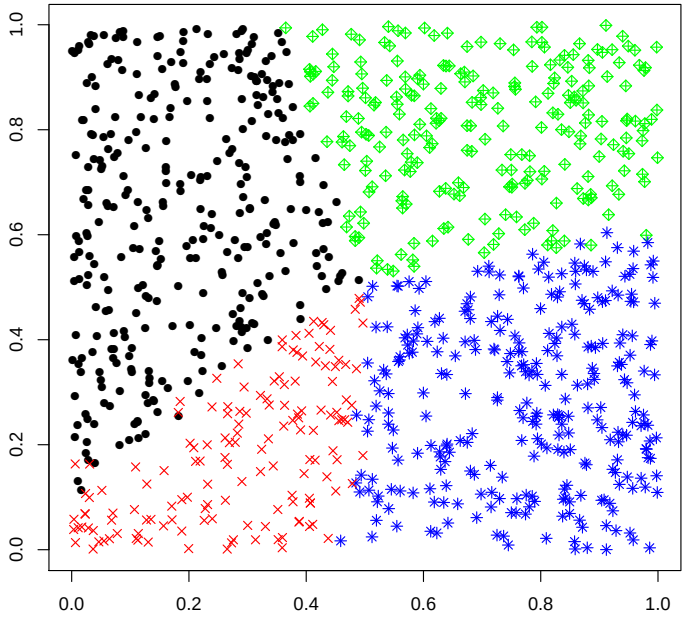
LDA groups the data into distinct classes. There is no way to tell that the true classes were assigned completely at random.

Now estimate the misclassification rate with cross-validation (LOOCV):

```
> n = length(yy[,3])
> pcv = rep(0, n)

> for (i in 1:n) {
+ zcv = lda(yy$yclass[-i] ~ ., yy[-i,1:2], prior = c(.25, .25, .25, .25))
+ ppcv = predict(zcv, yy[i,1:2])
+ pcv[i] = ppcv$class
+ }

> table(yy[,3], pcv)
   pcv
     1  2  3  4
  1 35 83 55 77
  2 41 80 58 71
  3 34 61 69 86
  4 38 69 58 85
```

Estimated Misclassification Rate: $731/1000 = 0.731$

The true misclassification rate is 0.75. So cross-validation estimates the misclassification rate well. Here is a plot showing the classifications assigned by cross-validation.



The class assignments made by cross-validation are almost identical to those made using the whole data set.

7.6 Quadratic Discriminant Analysis

The quadratic discriminant analysis (QDA) classifier is the Bayes classifier when populations are multivariate normal, with possibly different covariance matrices. Assume we have groups $\pi_1, ..., \pi_g$ that are distributed multivariate normal, with π_k having mean $\boldsymbol{\mu}_k$ and covariance matrix $\boldsymbol{\Sigma}_k$. Note that the covariance matrices are not assumed to be equal. Let $f_k(\mathbf{x})$ be the pdf and let p_k be the prior probability for group π_k. The QDA classifier therefore assigns $\mathbf{x}$ to the group for which $f_k(\mathbf{x})p_k$ is a maximum.

The pdf for group π_k is

$$f_k(\mathbf{x}) = (2\pi)^{-n/2}|\mathbf{\Sigma}_k|^{-1/2}\exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T\mathbf{\Sigma}_k^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)\right]$$

The constant $(2\pi)^{-n/2}$ does not depend on k. Dropping this constant and taking logs yields

$$-\frac{1}{2}\log|\mathbf{\Sigma}_k| - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T\mathbf{\Sigma}_k^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)$$

Adding the log of the prior and multiplying by -2 shows that the QDA classifier therefore assigns $\mathbf{x}$ to the group π_k for which

$$\log|\mathbf{\Sigma}_k| + (\mathbf{x}-\boldsymbol{\mu}_k)^T\mathbf{\Sigma}_k^{-1}(\mathbf{x}-\boldsymbol{\mu}_k) - 2\log p_k$$

is a *minimum*. In practice, $\boldsymbol{\mu}_1$, ..., $\boldsymbol{\mu}_g$ and $\mathbf{\Sigma}_1$ ..., $\mathbf{\Sigma}_g$ are unknown, and are replaced with their sample counterparts.

Is the Quadratic Discriminant Analysis Classifier Really Quadratic?
Yes. Multiplying out the part of the classifier that involves the item $\mathbf{x}$ to be classified yields

$$(\mathbf{x}-\boldsymbol{\mu}_k)^T\mathbf{\Sigma}_k^{-1}(\mathbf{x}-\boldsymbol{\mu}_k) = \mathbf{x}^T\mathbf{\Sigma}_k^{-1}\mathbf{x} - 2\boldsymbol{\mu}_k^T\mathbf{\Sigma}_k^{-1}\mathbf{x} + \boldsymbol{\mu}_k^T\mathbf{\Sigma}_k^{-1}\boldsymbol{\mu}_k$$

The term $\mathbf{x}^T\mathbf{\Sigma}_k^{-1}\mathbf{x}$ is quadratic in $\mathbf{x}$. In LDA, the matrix is $\mathbf{\Sigma}$, which is assumed to be the same for all groups, and thus does not depend on k. Therefore this term has no effect on the LDA classifier. In the QDA classifier, the matrix $\mathbf{\Sigma}_k$ is specific to group π_k, so the quadratic term depends on k.

Example 7.10. Samples of 50 American and 50 Canadian salmon have the diameter of their growth rings measured during a year of growth in fresh water and also for a year of growth in the ocean. It is desired to formulate a classification rule for classifying salmon as American (Group 1) or Canadian (Group 2). Following are the results:

Group	Fresh	Marine	Group	Fresh	Marine	Group	Fresh	Marine	Group	Fresh	Marine
1	108	368	1	85	444	2	129	420	2	134	383
1	131	355	1	109	397	2	148	371	2	117	355
1	105	469	1	106	442	2	179	407	2	126	345
1	86	506	1	82	431	2	152	381	2	118	379
1	99	402	1	118	381	2	166	377	2	120	369
1	87	423	1	105	388	2	124	389	2	153	403
1	94	440	1	121	403	2	156	419	2	150	354
1	117	489	1	85	451	2	131	345	2	154	390
1	79	432	1	83	453	2	140	362	2	155	349
1	99	403	1	53	427	2	144	345	2	109	325
1	114	428	1	95	411	2	149	393	2	117	344
1	123	372	1	76	442	2	108	330	2	128	400
1	123	372	1	95	426	2	135	355	2	144	403
1	109	420	1	87	402	2	170	386	2	163	370
1	112	394	1	70	397	2	152	301	2	145	355
1	104	407	1	84	511	2	153	397	2	133	375
1	111	422	1	91	469	2	152	301	2	128	383
1	126	423	1	74	451	2	136	438	2	123	349
1	105	434	1	101	474	2	122	306	2	144	373
1	119	474	1	80	398	2	148	383	2	140	388
1	114	396	1	95	433	2	90	385	2	150	339
1	100	470	1	92	404	2	145	337	2	124	341
1	84	399	1	99	481	2	123	364	2	125	346
1	102	429	1	94	491	2	145	376	2	153	352
1	101	469	1	87	480	2	115	354	2	108	339

Estimate the misclassification rate for both LDA and QDA, using LOOCV. Which rule has the smaller misclassification rate?

Solution: First we estimate the misclassification rate for LDA:

```
> library{MASS}
> pcv = rep(0, nrow(salmon))
> for (i in 1:nrow(salmon)) {
+ cvtrain = salmon[-i,]
+ zcv = lda(cvtrain$Group~., prior=c(0.5, 0.5), cvtrain)
+ ppcv = predict(zcv, salmon[i,])
+ pcv[i] = ppcv$class
+ }
> table(salmon$Group, pcv)
   pcv
     1  2
  1 44  6
  2  1 49
```

The misclassification rate for LDA is estimated to be 7/100, or 0.07. Now we estimate the misclassification rate for QDA:

```
> pcv = rep(0, nrow(salmon))
> for (i in 1:nrow(salmon)) {
+ cvtrain = salmon[-i,]
+ zcv = qda(cvtrain$Group~., prior=c(0.5, 0.5), cvtrain)
+ ppcv = predict(zcv, salmon[i,])
+ pcv[i] = ppcv$class
+ }
> table(salmon$Group, pcv)
   pcv
     1  2
  1 45  5
  2  3 47
```

The misclassification rate for QDA is estimated to be 8/100, or 0.08.

The misclassification for QDA is estimated to be 0.08. The misclassification for LDA was estimated to be 0.07. The rates for LDA and QDA appear to be about equal.

Comparing LDA and QDA

Both LDA and QDA estimate a multivariate normal distribution for each group. LDA requires that the covariance be the same for each group, while QDA allows the groups to have different covariances. Since QDA considers a wider range of models than LDA, the QDA estimates of the distributions are likely to have greater variance but smaller bias than those of LDA.

Specifically, let g be the number of groups, and let p be the dimension of $\mathbf{x}$. LDA estimates a separate mean for each group, but only one covariance matrix. This comes to a total of gp parameters for the mean plus $p(p+1)/2$ parameters for the covariance matrix. By comparison, since QDA estimates a separate covariance matrix for each group, a total of $gp + gp(p+1)/2$ parameters must be estimated.

When the training set is small, the variance of the QDA estimates can be very large, and LDA is likely to be a better choice. When there are enough items in the training set to estimate the many QDA parameters with reasonable precision, the variance will be less, and QDA is often the better choice.

7.7 Logistic Regression

Logistic regression is a method for estimating the probability that an item belongs to a given class as a function of variables. Specifically, we observe a vector $\mathbf{x} = (x_1, ..., x_p)^T$ on individuals in two classes. We will label the classes 0 and 1. Given $\mathbf{x}$, assume that the probability that an individual whose covariate values $\mathbf{x}$ belongs to class 1 is

$$p = \frac{\exp(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p)}{1 + \exp(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p)} = \frac{\exp(\boldsymbol{\beta}^T \mathbf{x})}{1 + \exp(\boldsymbol{\beta}^T \mathbf{x})}$$

Note that p will always be between 0 and 1, as required, no matter what the values of $\mathbf{x}$ and $\boldsymbol{\beta}$. This can be written as a linear model as follows:

$$\log \frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

The quantity $\log \dfrac{p}{1-p}$ is called the *logit* of p. As p varies from 0 to 1, the logit varies from $-\infty$ to ∞.

Assume we have observations on a training set of n individuals $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)$, where $\mathbf{x}_i$ is the vector of covariates on the ith individual and y_i is the class that individual i belongs to: either 0 or 1. We can estimate $\beta_0, ..., \beta_p$ using the method of logistic regression, which is simply a special case of maximum likelihood. We now describe this method.

The values $y_1, ..., y_n$ are Bernoulli trials with

$$P(y_i = 1) = \frac{\exp(\boldsymbol{\beta}^T \mathbf{x}_i)}{1 + \exp(\boldsymbol{\beta}^T \mathbf{x}_i)} \quad \text{and} \quad P(y_i = 0) = \frac{1}{1 + \exp(\boldsymbol{\beta}^T \mathbf{x}_i)}$$

The likelihood function is

$$L(\boldsymbol{\beta}) = \prod_{y_i=1} \frac{\exp(\boldsymbol{\beta}^T \mathbf{x}_i)}{1 + \exp(\boldsymbol{\beta}^T \mathbf{x}_i)} \prod_{y_i=0} \frac{1}{1 + \exp(\boldsymbol{\beta}^T \mathbf{x}_i)}$$

The maximum likelihood estimate $\boldsymbol{\beta}$ must be computed numerically.

Using logistic regression for classification

Let $\mathbf{x}$ be an item to be classified. Let y be the unknown class, 0 or 1, to which $\mathbf{x}$ belongs. Let $\hat{\boldsymbol{\beta}}$ be the MLE of $\boldsymbol{\beta}$. Let $p = P(y = 1)$, We estimate p with

$$\hat{p} = \frac{\exp(\hat{\boldsymbol{\beta}}^T \mathbf{x}_i)}{1 + \exp(\hat{\boldsymbol{\beta}}^T \mathbf{x}_i)}$$

If $\hat{p} \geq 0.5$, or equivalently, if $\log \dfrac{\hat{p}}{1+\hat{p}} \geq 0$, we classify $\mathbf{x}$ in class 1. Otherwise we classify $\mathbf{x}$ in class 0.

Example 7.11. The National Health Examination Survey studied 376 men with prostate cancer. The outcome (capsule) was whether the tumor had penetrated the prostratic capsule (class 1) or not (class 0). Several explanatory variables were measured; we consider two: The level of prostate specific antigen (PSA) in mg/ml, and tumor volume in cm^3. Following are the results of fitting a logistic regression, using R.

```
> pr1 = glm(capsule~psa+vol, family=binomial)
> summary(pr1)

Call:
glm(formula = capsule ~ psa + vol, family = binomial)

             Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.862942   0.183637  -4.699 2.61e-06
psa          0.049729   0.009340   5.324 1.01e-07
vol         -0.015944   0.006585  -2.421   0.0155

    Null deviance: 506.59  on 375  degrees of freedom
Residual deviance: 454.35  on 373  degrees of freedom
AIC: 460.35
```

A man with prostate cancer has a PSA level of 15 mg/ml and a tumor volume of 2 cm^3. How would he be classified?

Solution:
The logit is $\log \frac{\hat{p}}{1-\hat{p}} = \hat{\beta}_0 + \hat{\beta}_1 * \text{PSA} + \hat{\beta}_2 * \text{Volume} = -0.862942 + 0.049729(15) - 0.015944(2) = -0.1489$. Since this is less than 0, we classify the man in class 0. We predict that the tumor has not penetrated the prostratic capsule.

Logistic Classification with More than Two Populations
Assume we have K populations, and that we have observations on a training set of n individuals $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)$, where $\mathbf{x}_i = (x_{i1}, ..., x_{ip})^T$ is the vector of covariates on the ith individual and y_i is the class that individual i belongs to: an integer between 1 and K. Let p_{ij} be the probability that an individual with covariates $\mathbf{x}_i$ belongs to population j.

We must choose one population as the baseline population; we will choose population 1. We then model the log odds of the other outcomes as linear. Specifically, for $j = 2, ..., K$ we model

$$\log \frac{p_{ij}}{p_{i1}} = x_{i1}\beta_{j1} + \cdots + x_{ip}\beta_{jp}$$

Note that the parameters are different for each outcome j; we will let $\boldsymbol{\beta}_j$ denote the vector of parameters for outcome j. We now solve for p_{ij}. It follows that $p_{ij} = p_{i1}\exp(\mathbf{x}_i^T\boldsymbol{\beta}_j)$. Summing over j, we have

$$\sum_{j=2}^{K} p_{ij} = p_{i1}\sum_{j=2}^{K}\exp(\mathbf{x}_i^T\boldsymbol{\beta}_j)$$

Because the p_{ij} are constrained to sum to 1, we have

$$1 - p_{i1} = p_{i1}\sum_{j=2}^{K}\exp(\mathbf{x}_i^T\boldsymbol{\beta}_j)$$

Solving for p_{i1} yields

$$p_{i1} = \frac{1}{1+\sum_{j=2}^{K}\exp(\mathbf{x}_i^T\boldsymbol{\beta}_j)}$$

Therefore

$$p_{ij} = \frac{\exp(\mathbf{x}_i^T\boldsymbol{\beta}_j)}{1+\sum_{j=2}^{K}\exp(\mathbf{x}_i^T\boldsymbol{\beta}_j)}$$

The parameters $\boldsymbol{\beta}_2$, ..., $\boldsymbol{\beta}_K$ are estimated by maximum likelihood. To classify a new item with covariates $\mathbf{x}$, use the maximum likelihood estimates to compute estimates $\hat{p}_j$ for $j = 1, ..., K$ and assign the item to the population with the largest estimated probability.

Example 7.12. The 1996 American National Election Study conducted a survey in which individuals were asked about their political affiliation, along with other socioeconomic variables. The variables we will consider are

party: Democrat, Independent, or Republican

age: age in years

income: annual income

educ: Education level: Middle school, some high school, high school, some college

Fit a logistic model with age, income and education level as the explanatory variables, and predict the party affiliation of an individual aged 40 with a college education and an income of $50,000.

Solution:

We use the `multinom` command. This command is in the `nnet` library, because the maximization procedure uses a optimization method from the neural net trainer in `net`, but the procedure is otherwise unrelated to neural networks. The populations are D, I, and R. By default, D is chosen as the baseline because it is first in the alphabet.

```
> library(nnet}
> out = multinom(party~age+educ+income)
# weights:  30 (18 variable)
initial  value 1037.090001
iter  10 value 990.568608
iter  20 value 984.319052
final  value 984.166272
converged

> summary(out)
Call:
multinom(formula = party ~ age + educ + income)

Coefficients:
  (Intercept)          age educHSdrop    educHS   educColl educCCdeg educBAdeg
I   -1.373895 0.0001539014  0.2704482 0.2458744 0.09119446 0.3269554 0.1082654
R   -3.048576 0.0081945031  0.9876547 1.6915600 1.95336096 1.8835335 1.8708213
  educMAdeg     income
I 0.1933497 0.01623914
R 1.4539589 0.01724696

Std. Errors:
  (Intercept)         age educHSdrop    educHS educColl educCCdeg educBAdeg
I    0.766464 0.005374592  0.7460643 0.6992364 0.713322 0.7382877 0.7151263
R    1.111205 0.004902674  1.1237993 1.0721857 1.076931 1.0940829 1.0792352
  educMAdeg      income
I 0.7287344 0.003108590
R 1.0914118 0.002881749

Residual Deviance: 1968.333
AIC: 2004.333
```

Note that there are separate sets of parameters for Independents and Republicans. The parameters represent the log odds of being an Independent or Republican rather than a Democrat.

To classify an individual aged 40 with a college education and an income of $50,000, we'll compute the log odds of that person being an Independent or a Republican rather than a Democrat. The log odds of being an Independent rather than a Democrat are

$$-1.373895 + 0.0001539014(40) + 0.09119446 + 0.01623914(50) = -0.4646$$

Because the log odds are negative, we estimate that the probability that the person is an Independent is less than the probability that the person is a Democrat.

The log odds of being a Republican rather than a Democrat are

$$-3.048576 + .0081945301(40) + 1.95336096 + 0.01724696(50) = 0.0949$$

Because the log odds are positive, we estimate that the probability that the person is an Republican is greater than the probability that the person is a Democrat. We therefore classify the person as a Republican.

Example 7.13. The `iris` data set contains measurements of sepal length, sepal width, petal length, and petal length for 150 irises, composed of 50 plants in each of three species. A new data set contains information on 10 plants, whose species are unknown.

Use logistic regression to classify the new iris data on the basis of sepal length and sepal width, using the 50 plants of each species as a training set. Estimate the misclassification rate using LOOCV. The training set is in a file called `iris` and the data on the new plants is in a file called `newiris`.

Solution: We use R:

```
> lr = multinom(iris$Species~., iris[,1:2], trace=FALSE)
> coe = summary(lr)$coefficients

#Look at the coefficients
> coe
           (Intercept) Sepal.Length Sepal.Width
versicolor   -92.09924     40.40326   -40.58755
virginica   -105.10096     42.30094   -40.18799

> niris = t(newiris[,1:2])
> logodds = coe[,1] + coe[,2:3]%*%niris
> logodds = cbind(0, t(logodds))
> class = apply(logodds, 1, which.max)
> class
 [1] 2 1 1 2 3 2 3 2 3 3

#1 is setosa, 2 is versicolor, 3 is virginica
```

Now we'll estimate the misclassification rate with LOOCV.

```
> n = length(iris$Species)
> pcv = rep(0,n)
```

```
> for (i in 1:n) {
+ lr = multinom(iris$Species[-i]~., iris[-i,1:2], trace=FALSE)
+ coe = summary(lr)$coefficients
+ tiris = t(iris[,1:2])
+ logodds = coe[,1] + coe[,2:3]%*%tiris[,i]
+ logodds = cbind(0, t(logodds))
+ pcv[i] = which.max(logodds)
+ }

#Confusion Matrix
> table(iris$Species, pcv)
            pcv
               1  2  3
  setosa      49  1  0
  versicolor   0 36 14
  virginica    0 14 36

Misclassification Rate: 29/150 = 0.1933
```

Example 7.14. Use logistic regression to classify the iris data on the basis of sepal length, sepal width, petal length, and petal width. Estimate the misclassification rate using LOOCV.

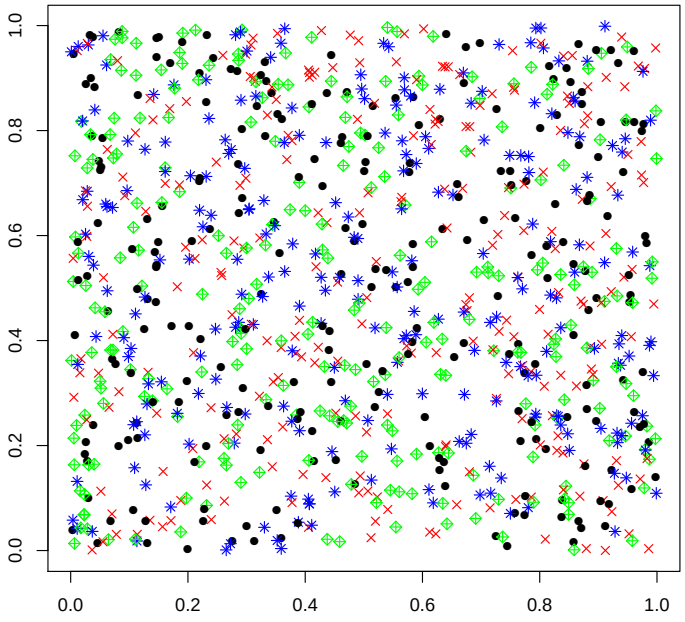
Solution: We use R:

```
> lr = multinom(iris$Species~., iris[,1:4], trace=FALSE)
> coe = summary(lr)$coefficients

#Look at the coefficients
> coe
           (Intercept) Sepal.Length Sepal.Width Petal.Length Petal.Width
versicolor    18.69037    -5.458424   -8.707401     14.24477   -3.097684
virginica    -23.83628    -7.923634  -15.370769     23.65978   15.135301

> niris = t(newiris[,1:4])
> logodds = coe[,1] + coe[,2:5]%*%niris
> logodds = cbind(0, t(logodds))
> class = apply(logodds, 1, which.max)
> class
 [1] 1 1 1 2 2 2 2 3 3 2

#1 is setosa, 2 is versicolor, 3 is virginica
```

Now we'll estimate the misclassification rate with LOOCV.

```
 n = length(iris$Species)
 pcv = rep(0,n)

 for (i in 1:n) {
 lr = multinom(iris$Species[-i]~., iris[-i,1:4], trace=FALSE)
 coe = summary(lr)$coefficients
 tiris = t(iris[,1:4])
 logodds = coe[,1] + coe[,2:5]%*%tiris[,i]
 logodds = cbind(0, t(logodds))
 pcv[i] = which.max(logodds)
 }

#Confusion Matrix
> table(iris$Species, pcv)
            pcv
              1  2  3
  setosa     50  0  0
  versicolor  0 48  2
  virginica   0  1 49

Misclassification Rate: 3/150 = 0.02
```

Example 7.15. The following plot presents 1000 points uniformly distributed on the unit square. The points have been randomly divided into four groups of 250 points each, as indicated by the different symbols on the plot. Fifty points from each group have been chosen at random to form a test set. Use logistic regression to classify the remaining points. Compute the misclassification rate on the test set, and use cross-validation to estimate the misclassification rate.



Solution: We use R:

```
#yy = 1000 uniform points on the unit square.
#yclass = true class for each point.
#train is the training set consisting of 50 points in each group.

> lr = multinom(yclass~., yy, trace=FALSE)

#Look at the coefficients

> coe = summary(lr)$coefficients
> coe
  (Intercept)          Y1          Y2
2 -0.11988471  0.2547843 -0.01825259
3 -0.29890181  0.1615529  0.42811822
4 -0.03676057 -0.2560816  0.31747226

> tyy = t(yy)
> logodds = coe[,1] + coe[,2:3]%*%tyy
> logodds = cbind(0, t(logodds))
> class = apply(logodds, 1, which.max)
```

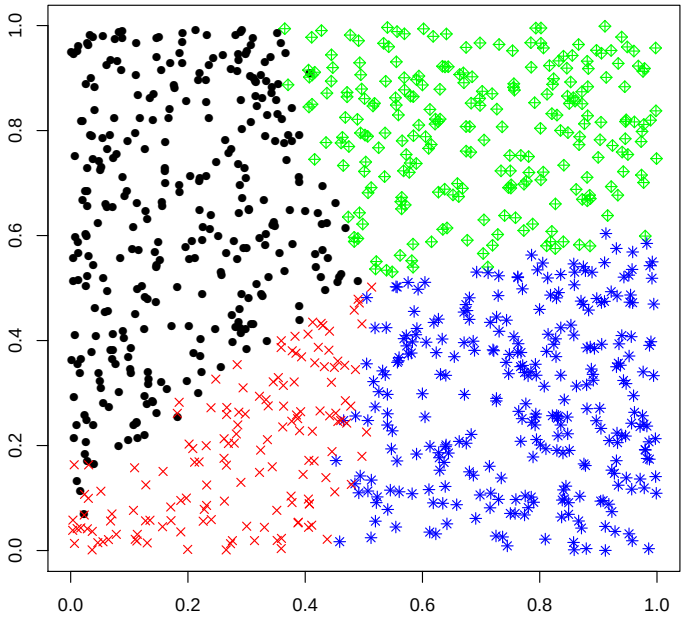
```
#Confusion Matrix
> table(yclass, class)
   class
     1  2  3  4
  1 39 80 56 75
  2 40 83 56 71
  3 34 60 70 86
  4 33 69 58 90

Misclassification rate: 718/1000 = 0.718
```

Now we'll use cross-validation to estimate the misclassification rate:

```
 n = length(yclass)
 pcv = rep(0, n)

 for (i in 1:n) {
 lr = multinom(yclass[-i]~., yy[-i,], trace=FALSE)
 coe = summary(lr)$coefficients
 logodds = coe[,1] + coe[,2:3]%*%t(yy[i,])
 logodds = cbind(0, t(logodds))
 pcv[i] = which.max(logodds)
 }

> table(yclass, pcv)
      pcv
yclass  1  2  3  4
     1 32 84 56 78
     2 43 77 59 71
     3 34 61 67 88
     4 38 69 60 83

Misclassification rate: 741/1000 = 0.741
```

The true misclassification rate is 0.75. Cross-validation estimates the misclassification rate well.

The following plot shows the classifications:



The classification is similar to that of LDA. The data are grouped into distinct classes. There is no way to tell from the plot that the true classes were assigned completely at random.

7.8 k Nearest Neighbors Classification

Most methods of classification rely on the assumption that items whose covariate values are closer together are more likely to be in the same class. The k-nearest neighbors method uses this assumption explicitly. The method is non-parametric, requiring no assumptions about the distributions of the covariates or the probabilities of class membership.

Assume we have observed a training set of n items. Each item belongs to one of M classes. For each item i we observe covariates $\mathbf{x}_i = (\mathbf{x}_{i1}, ..., \mathbf{x}_{ip})$, and class identifier y_i, where y_i is an integer between 1 and M.

The method for classifying new items is very simple. Begin by choosing a positive integer k. Let $\mathbf{x}$ be the vector of covariates of an item to be classified. The distance between $\mathbf{x}$ and an item in the training set $\mathbf{x}_i$ is simply the Euclidean distance between $\mathbf{x}$ and $\mathbf{x}_i$: $d_i = ||\mathbf{x} - \mathbf{x}_i||$. Identify the k members of the training set closest to the new item. The new item is assigned

to the class most frequently observed among these k nearest neighbors.

There are some variations on this procedure. For example, one may require a majority of the votes of the neighbors, rather than a simple plurality, to be classified. There are also a variety of rules for handling tie votes, and for handling the case of tied distances. We will not explore these ideas here.

The k-nearest neighbor method often works well in practice, so long as the assumption that items close in distance are likely to be in the same class is valid. One limitation of the method is that the search for the k nearest neighbors can be computationally intensive when there is a lot of data.

Example 7.16. The `iris` data set contains measurements of sepal length, sepal width, petal length, and petal length for 150 irises, composed of 50 plants in each of three species. A new data set contains information on 10 plants, whose species are unknown.

Use k-nearest neighbors to classify the new iris data on the basis of sepal length and sepal width, using the 50 plants of each species as a training set. Use $k = 1$ and $k = 3$. Estimate the misclassification rate using LOOCV for each value of k. The training set is in a file called `iris` and the data on the new plants is in a file called `newiris`.

Solution: We use R:

```
#Use knn command (load class library if needed)
#Columns 1 and 2 are sepal length and width. Column 5 is the species.

#k=1

> cls = knn(iris[, 1:2], newiris[,1:2], cl = iris[train,5], k=1)
> cls
 [1] versicolor setosa setosa setosa virginica  versicolor virginica
 [8] versicolor virginica versicolor
Levels: setosa versicolor virginica

#Estimate the misclassification rate using cross-validation

> n = nrow(iris)
> pre = rep(0, n)
> for (i in 1:n) {
+ pre[i] = knn(iris[-i, 1:2], iris[i,1:2], iris[-i, 5], k=1)
+ }
```

```
> table(pre, iris[,5])

pre setosa versicolor virginica
  1     49          0         0
  2      1         25        22
  3      0         25        28

48 of 150 items were misclassified: 32%

#Now try k=3

> cls = knn(iris[, 1:2], newiris[,1:2], cl = iris[train,5], k=1)
> cls
  [1] versicolor setosa setosa versicolor virginica versicolor virginica
  [8] versicolor virginica virginica
Levels: setosa versicolor virginica

#Estimate the misclassification rate using cross-validation

> n = nrow(iris)
> pre = rep(0, n)
> for (i in 1:n) {
+ pre[i] = knn(iris[-i, 1:2], iris[i,1:2], iris[-i, 5], k=3)
+ }
> table(pre, iris[,5])

pre setosa versicolor virginica
  1     49          0         0
  2      1         33        18
  3      0         17        32

36 of 150 items were misclassified: 24%
```

We see that $k = 3$ appears to perform better than $k = 1$.

Example 7.17. Use the training set to classify the new iris data, using values of k from 1 to 50. Use 10-fold cross-validation to estimate the misclassification rate for each value of k.

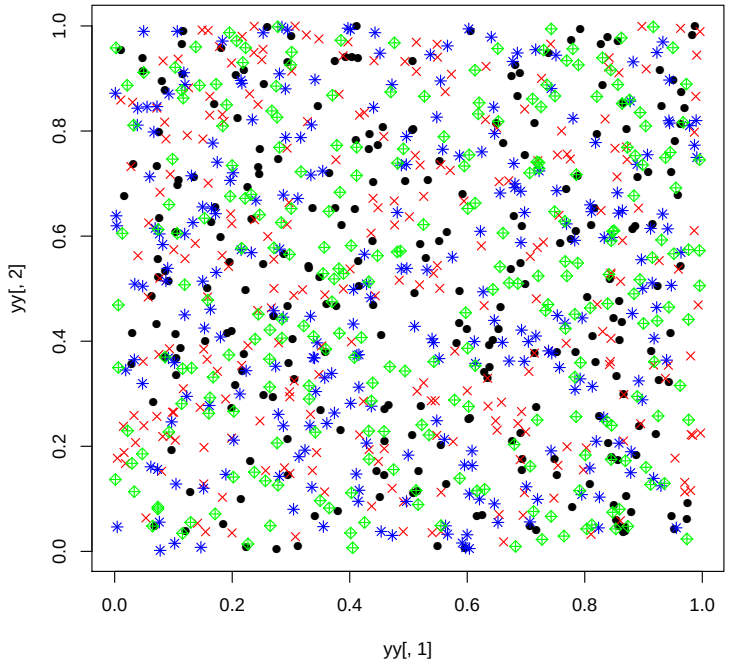
Solution:

```
> misclassrate = rep(0, 50)   #Initialize the misclassification rate for each k
> se = rep(0, 50)             #Initialize the standard error for each k
> nfold = rep(0, 10)          #nfold[i] is the number of items in fold i
> for (i in 1:10) {
> nfold[i] = length(fold[fold==i])
> }
```

```
> for (k in 1:50) {
> mcl = rep(0, 10)          #Initialize number of misclassfications for each fold.
> mclrate = rep(0, 10)      #Initialize misclassfication rate for each fold.
> for (i in 1:10) {
> pre = knn(iris[fold != i, 1:2], iris[fold==i,1:2], iris[fold !=i, 5], k)
> mcl[i] = sum(pre != iris[fold==i, "Species"])
> mclrate[i] = sum(pre != iris[fold==i, "Species"])/nfold[i]
> }
> se[k] = sd(mclrate)/sqrt(10)
> misclassrate[k] = sum(mcl)/n
> }

> round(misclassrate, 4)
 [1] 0.2733 0.2667 0.2333 0.2133 0.2067 0.2133 0.2400 0.2000 0.2067 0.2067 0.2200
[12] 0.2067 0.2067 0.2000 0.2200 0.2067 0.2200 0.2067 0.1933 0.2133 0.2067 0.2200
[23] 0.2133 0.2200 0.2200 0.2133 0.2200 0.2133 0.2067 0.2067 0.2067 0.2067 0.2000
[34] 0.2000 0.2000 0.2200 0.2200 0.2200 0.2200 0.2200 0.2267 0.2267 0.2200 0.2200
[45] 0.2200 0.2200 0.2267 0.2133 0.2267 0.2333
> round(se, 4)
 [1] 0.0323 0.0407 0.0313 0.0251 0.0238 0.0251 0.0248 0.0325 0.0362 0.0446 0.0502
[12] 0.0340 0.0468 0.0380 0.0398 0.0419 0.0476 0.0419 0.0428 0.0460 0.0476 0.0466
[23] 0.0452 0.0431 0.0431 0.0434 0.0356 0.0389 0.0379 0.0379 0.0379 0.0379 0.0381
[34] 0.0381 0.0411 0.0337 0.0337 0.0380 0.0388 0.0343 0.0322 0.0322 0.0395 0.0388
[45] 0.0432 0.0409 0.0360 0.0342 0.0360 0.0414
```

The minimum misclassification rate occurs at $k = 19$. However, the rates fluctuate up and down for values of k greater than 3 or 4, so it's not clear that there is any advantage to using such a large value of k. The values of the standard errors provide no evidence of an advantage to using a value of k greater than 3,

Example 7.18. The following plot presents 1000 points uniformly distributed on the unit square. The points have been randomly divided into four groups of 250 points each, as indicated by the different symbols on the plot. Use cross-validation to estimate the misclassification rate for k nearest neighbors with $k = 7$.

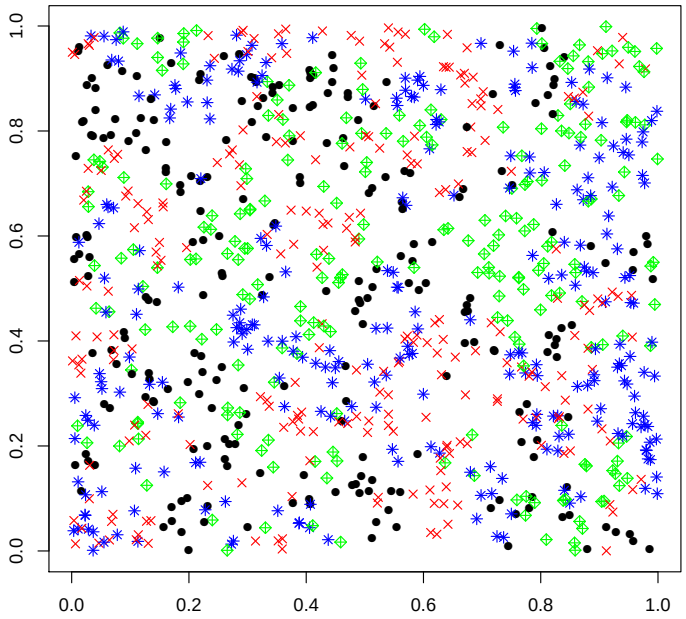


Solution: We use R:

```
#yy = 1000 uniform points on the unit square.

#yclass = true class for each point.

> n = 1000
> pre = rep(0,n)
> for (i in 1:n) {
> pre[i] = knn(yy[-i, 1:2], yy[i, 1:2], yclass[-i], k=7)
}

> table(pre, yyclass)
   yclass
pre  1  2  3  4
  1 60 70 62 58
  2 77 49 81 66
  3 58 65 42 63
  4 55 66 65 63

#786 of 1000 items misclassified: 78.6%
```

The true misclassification rate is 75%. Cross-validation estimates the rate fairly well.

The following plot shows the classifications:



The k-nearest neighbors has less tendency to define the classes as clumps than does LDA or logistic regression.

Some observations

- Classification methods are generally based on the assumption that points close together are likely to belong to the same class.
- The accuracy of any classification method, and of any method to estimate misclassification rates, relies on the assumption that the training set and test set are random samples from the same population, or equivalently, that the training and test sets are obtained by randomly dividing a larger set.
- Procedures to estimate the misclassification rate estimate the rate using the items in the training set. Therefore, if the training and test sets are not constructed as described above, which is often the case in practice, the misclassification rate will generally be underestimated.

- Proper use of classification methods, as with any method of data analysis, requires that judgment be used to interpret results in light of the fact that the assumptions of the method are invariably not precisely met in practice.

8 Support Vector Machines

8.1 The Maximal Margin Classifier

Support vector machines (SVMs) classify vectors in $\mathcal{R}^n$. We will focus on the case where there are two classes, which we denote -1 and 1. We have a training set $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)$ where $\mathbf{x}_i$ are vectors of predictor variables (features), and $y_i = 1$ or -1 is the class to which $\mathbf{x}_i$ belongs.

We consider the case in which the training set is linearly separable, which means that there is a hyperplane ($n-1$ dimensional linear surface) which separates the two classes, in that all the points with $y_i = 1$ are on one side of the hyperplane and all the points with $y_i = -1$ are on the other.

We will begin by focusing on the two-dimensional case. Figure 1 illustrates the situation. The *decision boundary* is the hyperplane, which is a line, separating the two classes. The distance from given point to the boundary is the length of a line drawn from the point perpendicular to the boundary. The *margin* is the minimum distance from a point to the boundary. The *maximal margin classifier* (also called the optimal separating hyperplane) is the line for which the margin is maximized. We will derive the maximal margin classifier.

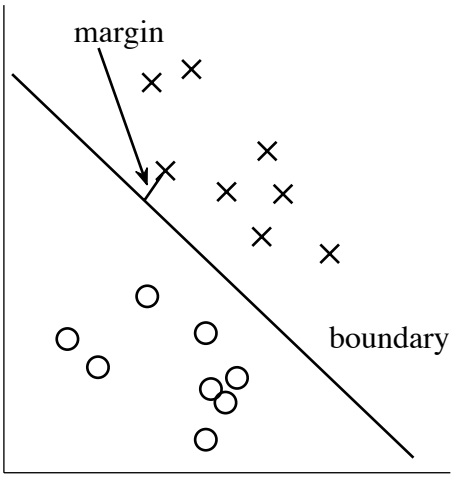


Figure 1. The two classes of points can be separated by a hyperplane (the boundary). The margin of the boundary is the minimum distance from the boundary to a point.

Figure 2 presents a boundary (line) and a vector $\mathbf{w}$ orthogonal to the line, drawn with the point $\mathbf{a}$ as its base. Let $\mathbf{x}$ be any point on the line.

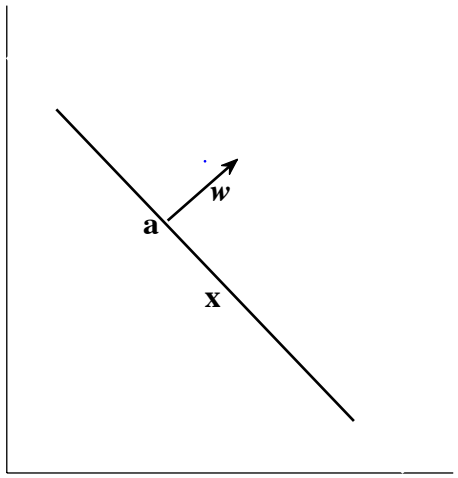


Figure 2. $\mathbf{w}$ is a vector whose base is at the point a on the boundary. x is any point on the boundary.

Theorem 8.1. The product $\mathbf{w}^T\mathbf{x}$ is constant for any point $\mathbf{x}$ on the boundary.

Proof:

Refer to Figure 2. The product $\mathbf{w}^T\mathbf{x} = \mathbf{w}^T\mathbf{a} + \mathbf{w}^T(\mathbf{x}-\mathbf{a})$. Since $\mathbf{w}^T(\mathbf{x}-\mathbf{a}) = 0$, $\mathbf{w}^T\mathbf{x} = \mathbf{w}^T\mathbf{a}$. Since $\mathbf{x}$ was arbitrary, $\mathbf{w}^T\mathbf{x}$ is the same for all $\mathbf{x}$.

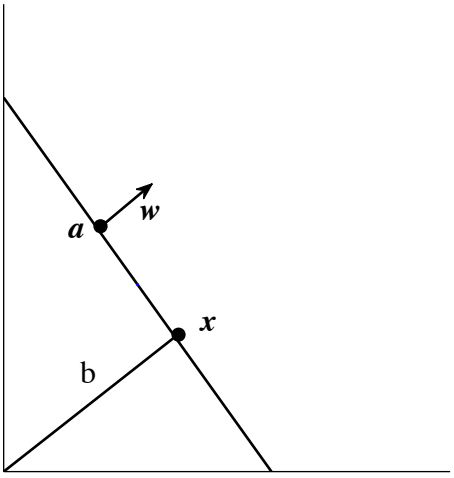


Figure 3. $\mathbf{w}$ is a unit vector ($||\mathbf{w}|| = 1$) orthogonal to the boundary and pointing away from the origin. $\mathbf{x}$ is any point on the boundary. b is the distance from the origin to the boundary.

Now we restrict $\mathbf{w}$ be be a unit vector ($||\mathbf{w}|| = 1$), orthogonal to the boundary. Let b be the distance from the origin to the boundary. The vector $\mathbf{w}$ and the number b define the boundary. $\mathbf{w}$ indicates the direction orthogonal to the boundary and b is the distance from the boundary to the origin. To see this, refer to Figure 3. The distance from the boundary to the origin is the length of the line from the origin perpendicular to the boundary. Let $\mathbf{x}$ be the point at which the line intersects the boundary, so the distance from the boundary to the origin is $||\mathbf{x}||$. Now the vector $\mathbf{x}$ points in the same direction as $\mathbf{w}$, so $\mathbf{w}^T\mathbf{x} = ||\mathbf{w}||\,||\mathbf{x}||\cos(0) = ||\mathbf{x}|| = b$. Therefore $b = \mathbf{w}^T\mathbf{x}$. We can see that the boundary consists of the points $\mathbf{x}$ such that

$$\mathbf{w}^T\mathbf{x} - b = 0$$

Figure 4 presents a boundary and a point $\mathbf{x}$ that is not on the boundary. Note that $\mathbf{x}$ is on the other side of the boundary from the origin. Let γ be the distance from $\mathbf{x}$ to the boundary. We derive an expression for γ.

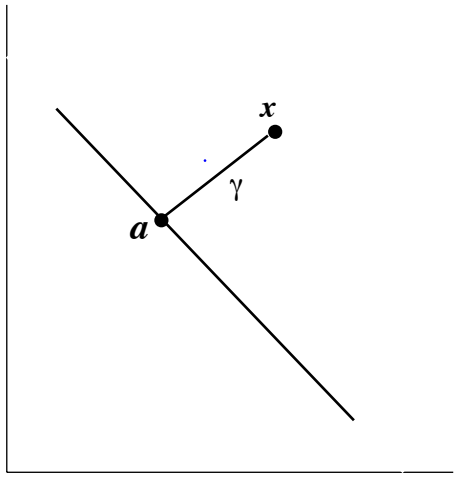


Figure 4. The distance from the boundary to the point $\mathbf{x}$ is γ.

Let $\mathbf{w}$ be a unit vector with its base on the boundary, orthogonal to the boundary, and pointing toward $\mathbf{x}$. Let $\mathbf{a}$ be the point on the boundary where the line perpendicular to the boundary through $\mathbf{x}$ intersects the boundary. Now $\mathbf{x} - \mathbf{a} = \gamma\mathbf{w}$, so

$$\mathbf{w}^T\mathbf{x} - \mathbf{w}^T\mathbf{a} = \gamma\mathbf{w}^T\mathbf{w}$$

Since $||\mathbf{w}|| = 1$, $\mathbf{w}^T\mathbf{w} = 1$. Since $\mathbf{a}$ is on the boundary, $\mathbf{w}^T\mathbf{a} = b$. Therefore

$$\gamma = \mathbf{w}^T\mathbf{x} - b$$

When $\mathbf{x}$ is on the other side of the boundary, $\gamma = -(\mathbf{w}^T\mathbf{x} - b)$. We now define the points on the side of the boundary in the direction of $\mathbf{w}$ to be in class 1, and the points on the other side to be in class -1. Let y be the class to which a point $\mathbf{x}$ belongs. Then

$$\gamma = y(\mathbf{w}^T\mathbf{x} - b)$$

Recall that the margin of a boundary is the distance to the closest point. Now imagine that we have a training set $\mathbf{x}_1, ..., \mathbf{x}_n$, and let y_i be the class to which $\mathbf{x}_i$ belongs ($y_i = \pm 1$). Assume the points are linearly separable, so that there exists a separating hyperplane. The maximal margin classifier is the boundary that maximizes the margin.

The boundary is determined by the values of $\mathbf{w}$ and b. To determine the marginal maximal classifier, we can solve the following maximization problem:

Find $\mathbf{w}, b$ maximizing γ, subject to the constraints $y_i(\mathbf{w}^T\mathbf{x}_i - b) \geq \gamma$ for all i, and $||\mathbf{w}|| = 1$.

The constraint $||\mathbf{w}|| = 1$ is non-convex, which makes the problem difficult to solve. We will rewrite the problem in a way that eliminates the constraint. First, we divide both sides of the inequality by γ.

Find $\mathbf{w}, b$ maximizing γ, subject to the constraints $y_i\left(\frac{\mathbf{w}^T\mathbf{x}_i}{\gamma} - \frac{b}{\gamma}\right) \geq 1$ for all i, and $||\mathbf{w}|| = 1$.

We haven't gotten rid of the $||\mathbf{w}|| = 1$ constraint yet. Now we'll get rid of it. Let $\mathbf{w}^* = \mathbf{w}/\gamma$. Since $||\mathbf{w}|| = 1$, $||\mathbf{w}^*|| = 1/\gamma$. Now maximizing γ is the same thing as minimizing $1/\gamma$, which is the same thing as minimizing $||\mathbf{w}^*||$. Finally, $b/\gamma = b||\mathbf{w}^*||$. The problem now becomes

Find $\mathbf{w}^*, b$ minimizing $||\mathbf{w}^*||$, subject to the constraints $y_i(\mathbf{w}^{*T}\mathbf{x}_i - b||\mathbf{w}^*||) \geq 1$ for all i.

Now some minor adjustments. Minimizing $||\mathbf{w}^*||$ is the same as minimizing $\frac{||\mathbf{w}^*||^2}{2}$. We will also replace $b||\mathbf{w}^*||$ with b. This will make the computations simpler later on. Also, we can replace $\mathbf{w}^*$ with $\mathbf{w}$. The final problem is

Find $\mathbf{w}, b$ minimizing $\frac{||\mathbf{w}||^2}{2}$, subject to the constraints $y_i(\mathbf{w}^T\mathbf{x}_i - b) \geq 1$ for all i.

The constraints imply that the points $\mathbf{x}_i$ closest to the boundary will have $y_i(\mathbf{w}^T\mathbf{x}_i - b) = 1$. These points are called *support vectors*. In the case we are now considering, where the two classes are linearly separable, the support vectors are the ones that lie on the margins. See Figure 5.

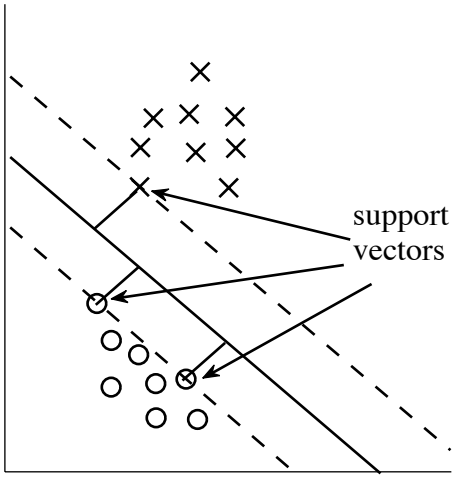


Figure 5. When two classes are linearly separable, the support vectors are the ones on the margins.

We will use the method of Lagrange multipliers for inequality constraints to solve this optimization problem. We give a brief description of the method.

8.2 Lagrange Multipliers for Inequality Constraints

Lagrange multipliers for inequality constraints extend the method of Lagrange multipliers to problems with inequality constraints. We will describe the method in this section, then in the next section we'll show how to apply the method to find maximum margin classifiers.

We first briefly review the method of Lagrange multipliers. Lagrange multipliers can be used to solve problems of the following form

Find $\mathbf{w}$ minimizing $f(\mathbf{w})$ subject to constraints $h_i(\mathbf{w}) = 0$, for $i = 1, , , m$.

We define the Lagrangian to be

$$L(\mathbf{w}, \boldsymbol{\beta}) = f(\mathbf{w}) + \sum_{i=1}^{m} \beta_i h_i(\mathbf{w})$$

Here $\beta_1, ..., \beta_m$ are called the Lagrange multipliers. We set the partial derivatives of L equal to 0:

$$\frac{\partial L}{\partial w_i} = 0 \qquad \frac{\partial L}{\partial \beta_i} = 0$$

The condition $\dfrac{\partial L}{\partial \beta_i} = 0$ ensures that the constraints $h_i(\mathbf{w}) = 0$ will be satisfied. The equations are solved for $\mathbf{w}$ and $\boldsymbol{\beta}$.

We now generalize this method to problems which contain inequality constraints. We give only the main ideas and results, without developing the theory in detail. We want to solve problems of the following form:

Minimize $f(\mathbf{w})$ subject to constraints $g_i(\mathbf{w}) \leq 0$ for $i = 1, ..., k$, and $h_i(\mathbf{w}) = 0$, for $i = 1, , , m$

We refer to this problem as the *primal* problem. To solve this problem we define the function called the *generalized Lagrangian*:

$$L(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{w}) + \sum_{i=1}^{k} \alpha_i g_i(\mathbf{w}) + \sum_{i=1}^{m} \beta_i h_i(\mathbf{w})$$

The α_i's and β_i's are the Lagrange multipliers. We constrain $\alpha_i \geq 0$. If we maximize this function over $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ we obtain a function of $\mathbf{w}$ called the *primal* function, which we denote θ_P.

$$\theta_P(\mathbf{w}) = \max_{\boldsymbol{\alpha}, \boldsymbol{\beta}:\, \alpha_i \geq 0} L(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \max_{\boldsymbol{\alpha}, \boldsymbol{\beta}:\, \alpha_i \geq 0} f(\mathbf{w}) + \sum_{i=1}^{k} \alpha_i g_i(\mathbf{w}) + \sum_{i=1}^{m} \beta_i h_i(\mathbf{w})$$

We now show that the value of the primal function is ∞ at any value of $\mathbf{w}$ that violates one or more of the constraints. Let $\mathbf{w}$ be a value for which $g_i(\mathbf{w}) > 0$ for some i. Then the term $\alpha_i g_[(\mathbf{w})$ can be made as large as desired by taking α_i sufficiently large. If $h_i(\mathbf{w}) \neq 0$ for some i, then $\beta_i h_i(\mathbf{w})$ can be made arbitrarily large by taking β_i to be sufficiently large (negative if $h_i(\mathbf{w}) < 0$, positive if $h_i(\mathbf{w}) > 0$).

Conversely, if $\mathbf{w}$ is a value for which the constraints are satisfied, then $\theta_P(\mathbf{w}) = f(\mathbf{w})$. To see this, note that $h_i(\mathbf{w}) = 0$ for all i, so $\sum_{i=1}^{m} \beta_i h_i(\mathbf{w}) = 0$. In addition, $g_i(\mathbf{w}) \leq 0$ for all i. Since α_i is constrained to be nonnegative, $\alpha_i g_i(\mathbf{w})$ is maximized for $\alpha_i = 0$, thus the maximum value of $\sum_{i=1}^{k} \alpha_i g_i(\mathbf{w})$ is 0. In summary, $\theta_P(\mathbf{w}) = f(\mathbf{w})$ for all $\mathbf{w}$ that satisfy the constraints, and $\theta_P(\mathbf{w}) = \infty$ for all $\mathbf{w}$ that do not satisfy the constraints.

The primal problem is to minimize the primal function. So the solution to the primal problem is

$$p^* = \min_{\boldsymbol{w}} \theta_P(\mathbf{w}) = \min_{\boldsymbol{w}} \max_{\boldsymbol{\alpha}, \boldsymbol{\beta}:\, \alpha_i \geq 0} L(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta})$$

Now we look at a problem called the *dual* problem. Define

$$\theta_D(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \min_{\boldsymbol{w}} L(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta})$$

θ_D is called the *dual* function. For the dual function θ_D we are minimizing over $\mathbf{w}$, whereas for the primal function θ_P we maximized over $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. The dual problem is to maximize the dual function:

$$\max_{\boldsymbol{\alpha}, \boldsymbol{\beta}: \alpha_i \geq 0} \theta_D(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \max_{\boldsymbol{\alpha}, \boldsymbol{\beta}: \alpha_i \geq 0} \min_{\boldsymbol{w}} L(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta})$$

Let $d^* = \max_{\boldsymbol{\alpha}, \boldsymbol{\beta}: \alpha_i \geq 0} \theta_D(\boldsymbol{\alpha}, \boldsymbol{\beta})$ be the optimal value of $\theta_D(\mathbf{w})$.

We can now see how the dual and primal problems are related.

$$d^* = \max_{\boldsymbol{\alpha}, \boldsymbol{\beta}: \alpha_i \geq 0} \min_{\boldsymbol{w}} L(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \qquad p^* = \min_{\boldsymbol{w}} \max_{\boldsymbol{\alpha}, \boldsymbol{\beta}: \alpha_i \geq 0} L(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta})$$

It turns out that $d^* = p^*$ under the following conditions:

- The Hessians of $f(\mathbf{w})$ and of each $g_i(\mathbf{w})$ for are nonnegative definite (i.e., $f(\mathbf{w})$ and $g_i(\mathbf{w})$ are convex)
- For each i, $h_i(\mathbf{w}) = \mathbf{a}_i^T \mathbf{w} + b_i$ for some vector $\mathbf{a}_i$ and real number b_i, i.e., $h(\mathbf{w})$ is affine.
- There exists a value of $\mathbf{w}$ for which $g_i(\mathbf{w}) < 0$ for all i, i.e., the constraints g_i are strictly feasible.

Under these conditions there exist $\mathbf{w}^*$, $\boldsymbol{\alpha}^*$, and $\boldsymbol{\beta}^*$ such that $p^* = d^* = L(\mathbf{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$. In addition, $\mathbf{w}^*$, $\boldsymbol{\alpha}^*$, and $\boldsymbol{\beta}^*$ satisfy the following conditions, known as the *Karush-Kuhn-Tucker (KKT) conditions*:

1. $\frac{\partial}{\partial w_i} L(\mathbf{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) = 0$ for all i.
2. $\frac{\partial}{\partial \beta_i} L(\mathbf{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) = 0$ for all i.
3. $\alpha_i^* g_i(\mathbf{w}^*) = 0$ for all i.
4. $g_i(\mathbf{w}^*) \leq 0$ for all i.
5. $\alpha_i^* \geq 0$ for all i.

It can be shown that if there exist $\mathbf{w}^*$, $\boldsymbol{\alpha}^*$, and $\boldsymbol{\beta}^*$ satisfying the KKT conditions, then $\mathbf{w}^*$, $\boldsymbol{\alpha}^*$, and $\boldsymbol{\beta}^*$ are solutions to both the primal and dual problems.

Condition 3 is of particular interest. It is called the *KKT dual complementarity condition*. It implies that if $\alpha_i > 0$ then $g(\mathbf{w}^*) = 0$. We will see that as a result, the classifier depends only on the support vectors.

8.3 Computing the Maximal Margin Classifier

We are ready to apply Lagrangian multipliers to compute the maximal margin classifier. We wish to perform the following optimization:

$$\text{Find } \mathbf{w}, b \text{ minimizing } \frac{||\mathbf{w}||^2}{2}, \text{ subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \text{ for all } i.$$

The inequality constraints are $g_i(\mathbf{w}) = 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)$. There are no equality constraints. The Lagrangian is

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{||\mathbf{w}||^2}{2} - \sum_{i=1}^{m} \alpha_i[y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1]$$

Note that the KKT dual complementarity condition (condition 3) ensures that $\alpha_i = 0$ unless $y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1$, which occurs only for the points exactly on the margin, that is, the support vectors.

We will construct the dual problem. To do this, we must minimize the Lagrangian with respect to $\mathbf{w}$ and b, by setting partial derivatives equal to 0. To make this easier, we rewrite the Lagrangian in terms of the components of the vectors $\mathbf{w}$ and $\mathbf{x}_i$.

$$\begin{aligned} L(\mathbf{w}, b, \boldsymbol{\alpha}) &= \frac{1}{2}\sum_{j=1}^{n} w_j^2 - \sum_{i=1}^{m} \alpha_i \left[y_i \left(\sum_{j=1}^{n} w_j x_{ij} + b \right) - 1 \right] \\ &= \frac{1}{2}\sum_{j=1}^{n} w_j^2 - \sum_{i=1}^{m}\sum_{j=1}^{n} \alpha_i y_i w_j x_{ij} + b\sum_{i=1}^{m}\sum_{j=1}^{n} \alpha_i y_i - \sum_{i=1}^{m} \alpha_i \end{aligned}$$

We set the partial derivatives of $L(\mathbf{w}, b, \boldsymbol{\alpha})$ equal to 0.

$$\frac{\partial L}{\partial w_k} = w_k - \sum_{i=1}^{m} \alpha_i y_i x_{ik} = 0 \quad \text{for } k = 1., ..., n$$

In vector form this is

$$\nabla_{\mathbf{w}} L = \mathbf{w} - \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i = \mathbf{0}$$

It follows that

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$$

The partial derivative with respect to b yields

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{m} \alpha_i y_i = 0$$

Substituting the expression for $\mathbf{w}$ into the Lagrangian yields

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

Simplifying and using fact that $\sum_{i=1}^{m} \alpha_i y_i = 0$ yields

$$D(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

The dual problem is to maximize $D(\boldsymbol{\alpha})$ subject to the previously stated constraints:

$$\text{Find } \boldsymbol{\alpha} \text{ maximizing } D(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$
$$\text{subject to } \alpha_i \geq 0 \text{ for all } i, \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0.$$

The dual problem must be solved numerically. The values of α_i can then be used to find the optimal $\mathbf{w}$, using the equation $\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$.

It can be shown that the optimal value of b is

$$b = -\frac{\max_{i:y_i=-1} \mathbf{w}^T \mathbf{x}_i + \min_{i:y_i=1} \mathbf{w}^T \mathbf{x}_i}{2}$$

Classifying a new point

Once we have computed $\mathbf{w}$ and b, we can use the points $\mathbf{x}_1, ..., \mathbf{x}_n$ to classify a new point $\mathbf{x}$.

$$\text{Classify } \mathbf{x} \text{ as } 1 \text{ if } \mathbf{w}^T \mathbf{x} + b > 0$$
$$\text{Classify } \mathbf{x} \text{ as } -1 \text{ if } \mathbf{w}^T \mathbf{x} + b < 0$$

Now $\mathbf{w}^T\mathbf{x} = \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i^T \mathbf{x}$, so we can express the classifier in the following useful form:

$$\text{Classify } \mathbf{x} \text{ as } 1 \text{ if } \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b > 0$$
$$\text{Classify } \mathbf{x} \text{ as } -1 \text{ if } \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b < 0$$

We can see that the classifier depends only on the inner products of $\mathbf{x}$ with the training set points $\mathbf{x}_i$. Furthermore, $\alpha_i \neq 0$ only for the support vectors, so the classifier depends only on the inner products of $\mathbf{x}$ with the support vectors.

8.4 Non-Separable Data Sets

Thus far we have assumed that there exists a linear boundary that will separate the two classes in our training set. This is not always the case. Even when a training set is linearly separable, it may be best to use a boundary that misclassifies a few items. Figure 6 illustrates such a case. The solid line separates the two classes, but the dashed line may well be the more accurate classifier, even though it misclassifies one of the items.

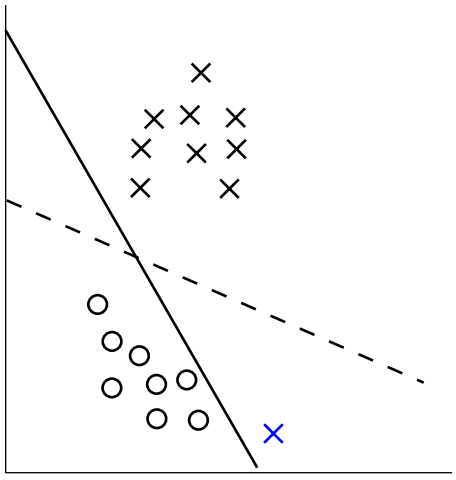


Figure 6. The solid line separates the two classes, but the dashed line may be the more accurate classifier, even though it misclassifies one of the items.

We now extend the definition of the maximal margin classifier to allow misclassification. We consider the case where the boundary is linear. The resulting classifier is called a *support vector classifier*.

Let $\mathbf{x}_1, ..., \mathbf{x}_m$ be vectors in a training set. To construct a support vector classifier, we specify a positive constant C and modify the optimization used for the maximal margin classifier as follows:

$$\text{Find } \mathbf{w}, b, \xi_i \text{ minimizing } \frac{||\mathbf{w}||^2}{2} + C\sum_{i=1}^{m} \xi_i$$
$$\text{subject to the constraints } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0, \text{ for } i = 1, ..., m.$$

To explain this new optimization problem, recall that for a point $\mathbf{x}_i$ that is on the margin, $y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1$ and for a point on the boundary, $y_i(\mathbf{w}^T\mathbf{x}_i + b) = 0$. The constraint $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$ allows points to be inside the margin (when $\xi_i > 0$. If $\xi_i > 1$, then $\mathbf{x}_i$ is misclassified.

The tuning parameter C is called the "cost." Choosing a larger value for C will tend to force the $\sum_{i=1}^{m} \xi_i$ to be smaller, so fewer points will be inside the margin or be misclassified.

We now solve the optimization problem. The Lagrangian is

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{||\mathbf{w}||^2}{2} + C\sum_{i=1}^{m} \xi_i - \sum_{i=1}^{m} \alpha_i[y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i] - \sum_{i=1}^{m} r_i\xi_i$$

The dual problem is constructed by minimizing the Lagrangian with respect to $\mathbf{w}$, b, and ξ_i by taking partial derivatives and setting them equal to 0. The dual problem is

$$\text{Find } \boldsymbol{\alpha} \text{ maximizing } D(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n} \alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$$
$$\text{subject to } 0 \leq \alpha_i \leq C \text{ for all } i, \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0.$$

The only change is that the constraints $\alpha_i \geq 0$ have been replaced with $0 \leq \alpha_i \leq C$.

The equations used to classify new points are the same as in the linearly separable case:

$$\text{Classify } \mathbf{x} \text{ as } 1 \text{ if } \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i^T\mathbf{x} + b > 0$$
$$\text{Classify } \mathbf{x} \text{ as } -1 \text{ if } \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i^T\mathbf{x} + b < 0$$

It can be shown that the KKT conditions imply

- If $y_i(\mathbf{w}^T\mathbf{x}_i + b) > 1$, then $\alpha_i = 0$
- If $y_i(\mathbf{w}^T\mathbf{x}_i + b) < 1$, then $\alpha_i = C$
- If $y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1$, then $0 \leq \alpha_i \leq C$

Therefore the classifier is determined only by points on or inside the margin. These are the support vectors.

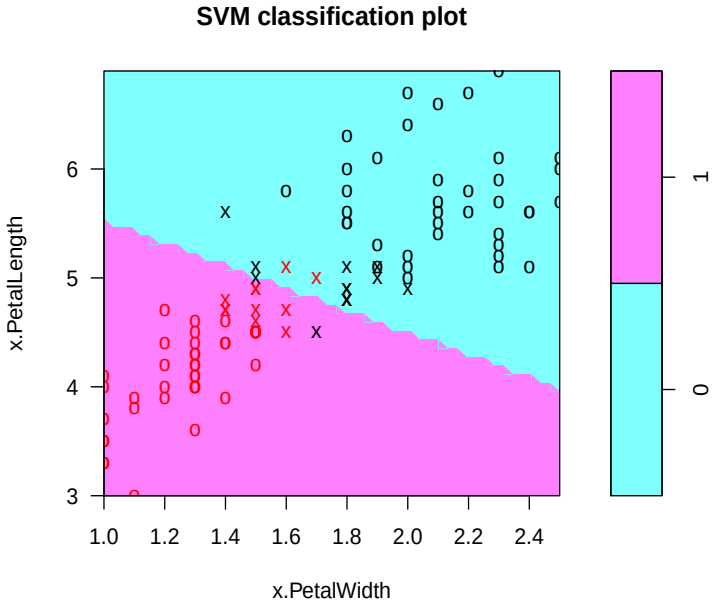
Example 8.1. We use a support vector machine to classify the iris data. We include only those observations classified as Versicolor or Virginia, since the third species (Setosa) is easily differentiated from the other two. We also use only the petal width and petal length. Here is a part of the data set.

Petal Length	Petal Width	Species
4.7	1.4	1
4.5	1.5	1
4.9	1.5	1
4.0	1.3	1
4.6	1.5	1
4.5	1.3	1
⋮	⋮	⋮
5.2	2.3	0
5.0	1.9	0
5.2	2.0	0
5.4	2.3	0
5.1	1.8	0

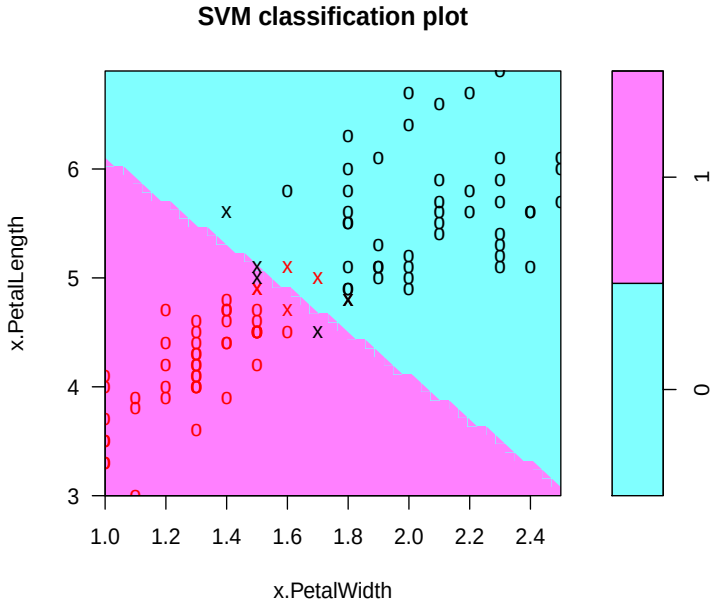
The `svm` command trains a support vector machine. We present a linear kernel (which specifies that the boundary will be linear) with a cost of 1. The option "scale = FALSE" specifies that the data should not be scaled.

```
> library(e1071)

> irisdat = data.frame(x=iris[,c(1,2)], y = as.factor(iris[,3]))
> svmfit=svm(y~., data=irisdat, kernel="linear", cost=1, scale=FALSE)

#We can plot the boundary.
```

```
> plot(svmfit, data=irisdat)

# We repeat the procedure for a cost of 100.

> svmfit=svm(y~., data=irisdat, kernel="linear", cost=100, scale=FALSE)
> plot(svmfit, data=irisdat)
```



Cost = 1



Cost = 100

The points labeled "x" are the support vectors, that is, the vectors that are inside the margin.. Unfortunately, the margin isn't shown. Note that the margin is narrower, and there are fewer support vectors, when the cost is higher. Points in black are in class 0 and those in red are in class 1. When cost = 1, either three or four points are misclassified; the boundary isn't drawn precisely enough to tell. When cost = 100, five points are misclassified.

8.5 Kernels

We have seen that the maximum margin classifier classifies a vector $\mathbf{x}$ on the basis of inner products $\mathbf{x}_i^T\mathbf{x}$, where the $\mathbf{x}_i$ are points in a training set. This produces a linear boundary between the classes. Often a non-linear boundary is more appropriate, so we would like to extend the method to construct non-linear boundaries. One way to do this would be to add components to $\mathbf{x}$ that are non-linear functions of the original components. For example, if

$\mathbf{x} = (x_1, x_2, x_3)$, we could replace $\mathbf{x}$ with $\phi(\mathbf{x}) = (x_1, x_2, x_3, x_1^2, x_2^2, x_3^2)$, and substitute $\phi(\mathbf{x})$ for $\mathbf{x}$ in the classifier, so the classification would be performed on the basis of the inner products $\phi(\mathbf{x}_i)^T\phi(\mathbf{x})$.

A function that maps a vector $\mathbf{x}$ to a (generally larger) vector $\phi(\mathbf{x})$ whose components are functions of the components of $\mathbf{x}$ is called a *feature mapping*. A function that maps pairs of vectors $(\mathbf{x}, \mathbf{z})$ to the inner product $\phi(\mathbf{x})^T\phi(\mathbf{z})$ is called a *kernel*. We now present formal definitions:

Definition 8.1. A *feature mapping* is a function ϕ that maps a vector $\mathbf{x}$ to a vector $\phi(\mathbf{x})$ whose components are functions of the components of $\mathbf{x}$.

Definition 8.2. Let $\phi(\mathbf{x})$ be a feature mapping. A *kernel* is a function $K(\mathbf{x}, \mathbf{z})$ that maps the pair of vectors $(\mathbf{x}, \mathbf{z})$ to the inner product $\phi(\mathbf{x})^T\phi(\mathbf{z})$, that is, $K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T\phi(\mathbf{z})$.

Kernels can often be expressed without explicitly describing the feature mapping. Following is an example.

Example 8.2. We show that the function $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z})^2$ is a kernel. We multiply out to obtain

$$\begin{aligned} K(\mathbf{x}, \mathbf{z}) &= (\mathbf{x}^T\mathbf{z})^2 \\ &= \left(\sum_{i=1}^{n} x_i z_i\right)^2 \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i z_i x_j z_j \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} (x_i x_j)(z_i z_j) \end{aligned}$$

Now define $\phi(\mathbf{x}) = (x_1x_1, x_1x_2, ..., x_1x_n, x_2x_1, ..., x_nx_1, ..., x_nx_n)$, a vector of dimension n^2 consisting of all products of pairs of components of $\mathbf{x}$. It is clear that $K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T\phi(\mathbf{z})$.

Note that computing the kernel in the form $(\mathbf{x}^T\mathbf{z})^2$ requires $O(n)$ operations, while computing the inner product $\phi(\mathbf{x})^T\phi(\mathbf{z})$ requires $O(n^2)$ operations. It is thus much more efficient

to compute the $K(\mathbf{x}, \mathbf{z})$ directly without using the feature mapping. Computing a kernel without explicitly computing $\phi(\mathbf{x})$ is known as the "kernel trick."

In the previous example, we showed that $K(\mathbf{x}, \mathbf{z})$ was a kernel by explicitly constructing the corresponding feature mapping ϕ. In general, this is not necessary. We need to define a matrix, called the *kernel matrix*.

Theorem 8.2. (Mercer's Theorem:) Let $K(\mathbf{x}, \mathbf{z})$ be a function mapping pairs of vectors to real numbers. Let $\mathbf{x}_1, ..., \mathbf{x}_n$ be a collection of n vectors. Consider the $n \times n$ matrix whose ij element is $K(\mathbf{x}_i, \mathbf{x}_j)$. Then there exists a feature mapping $\phi(\mathbf{x})$ such that $K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \phi(\mathbf{z})$ if and only if the matrix is symmetric and non-negative definite for any collection of vectors $\mathbf{x}_1, ..., \mathbf{x}_n$.

Proof: Omitted

Note that the feature mapping $\phi(\mathbf{x})$ may be infinite-dimensional.

Many kernels have been proposed for use with support vector machines. Here are the four kernels supported in R:

- Linear: $K(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T \mathbf{z}$
- Polynomial: $K(\mathbf{x}, \mathbf{z}) = (\gamma \mathbf{x}^T \mathbf{z} + c)^d$. The constants γ, c, and d must be specified by the user.
- Radial: $K(\mathbf{x}, \mathbf{z}) = \exp(-\gamma ||\mathbf{x} - \mathbf{z}||^2)$. The constant γ must be specified by the user.
- Sigmoid: $K(\mathbf{x}, \mathbf{x}) = \tanh(\gamma \mathbf{x}^T \mathbf{z} + c)$. The constants γ and c must be specified by the user.

We note that for the radial and sigmoid kernels, it turns out that the feature mapping ϕ is infinite-dimensional.

Classifying observations using a kernel

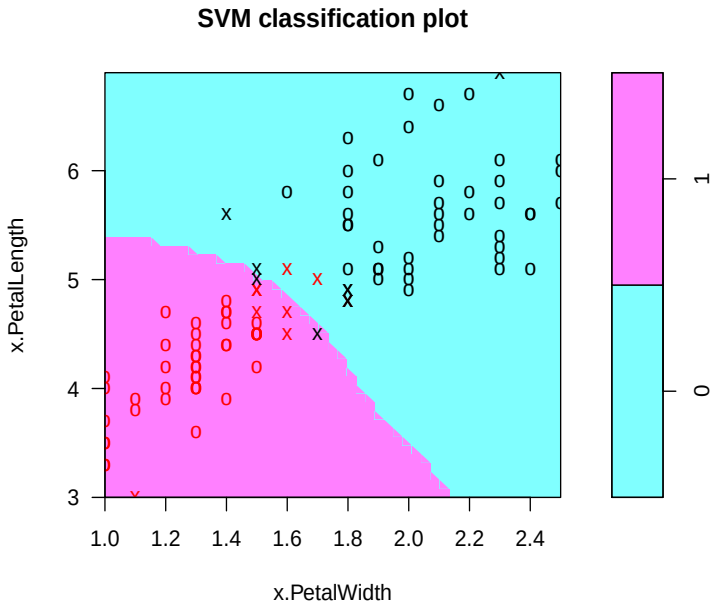
Let $\mathbf{x}_1, ..., \mathbf{x}_m$ be the vectors in a training set, and let $\mathbf{x}$ be the vector to be classified. Using a kernel is equivalent to replacing the vectors $\mathbf{x}_i$ and $\mathbf{x}$ with feature mappings $\phi(\mathbf{x}_i)$ and $\phi(\mathbf{x})$. We could then use the classification method above, replacing $\mathbf{x}_i^T \mathbf{x}$ with $\phi(\mathbf{x}_i)^T \phi(\mathbf{x})$. Of course, we do not actually need to compute or even know the feature mapping ϕ. We simply use the kernel $K(\mathbf{x}_i, \mathbf{x})$ instead. The classification rule becomes

$$\text{Classify } \mathbf{x} \text{ as 1 if } \sum_{i=1}^{m} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b > 0$$
$$\text{Classify } \mathbf{x} \text{ as -1 if } \sum_{i=1}^{m} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b < 0$$

Example 8.3. Use a radial kernel to classify the iris data on the basis of sepal length and sepal width.

With a radial kernel we must specify both the cost and the parameter γ. We'll specify cost $= 5$ and $\gamma = 1$.

```
> svmfit=svm(Species~., data=irisdat, kernel="radial", cost=5, gamma=1, scale=FALSE)
> plot(svmfit, data=irisdat)
```



SVM with a radial kernel. The boundary is curved.

Example 8.4. Use LOOCV to estimate the misclassification rate for the SVM with radial kernel, cost $= 5$ and gamma $= 1$.

```
> pre = rep(0, n)
> for (i in 1:n) {
+ svmfit = svm(Species~., irisdat[-i,], kernel="radial", cost=5, gamma=1, scale=FALSE)
+ pre[i] = predict(svmfit, irisdat[i,1:2])
+ }
```

```
> table(pre, irisdat$Species)

pre  0  1
  1 47  3
  2  3 47

#Estimated misclassification rate: 0.06
```

In practice, we choose the tuning parameters, (cost, gamma) by cross-validation. We split the data into a training set and a test set. We train support vector machines with various values of the parameters on the training set, then use them to classify the values in the test set. We choose the parameters that minimize the misclassification rate. The command `tune` makes this easy. This command uses 10-fold cross-validation to choose the best values for parameters from a list specified by the user. We'll find the best linear and best radial kernel, as chosen by 10-fold cross-validation, for the iris data.

```
> set.seed(1)
> tune.out = tune(svm, Species~., data=irisdat, kernel="linear",
+ ranges=list(cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100)))

Parameter tuning of svm:

- sampling method: 10-fold cross validation

- best parameters:
 cost
 0.01

- best performance: 0.06

- Detailed performance results:
   cost error dispersion
1 1e-03  0.61 0.22827858
2 1e-02  0.06 0.06992059
3 1e-01  0.06 0.06992059
4 1e+00  0.07 0.06749486
5 5e+00  0.07 0.06749486
6 1e+01  0.07 0.06749486
7 1e+02  0.07 0.06749486
```

The detailed performance results show the result of the cross-validation. For each cost, the error is the average of the misclassification rates over the 10 folds and the dispersion is the

standard deviation. We can see that costs of 0.01 and 0.1 performed equally well, with an estimated error rate of 0.06. Error rates for higher costs were only slightly higher, and well within the margin of error specified by the standard deviations.

Following are data on 10 iris plants whose species are known to be either versicolor or virginica.

Sepal Length	Sepal Width	Petal Length	Petal Width
7.0	3.4	4.5	1.3
8.0	3.5	4.0	1.2
7.2	3.1	3.9	2.7
4.8	1.7	2.8	1.2
6.7	4.0	4.6	1.3
5.9	1.7	3.1	2.1
7.4	2.7	4.6	1.2
6.0	3.6	5.5	1.7
7.7	2.3	4.7	2.0
6.8	3.8	4.1	2.0

Now we'll use the chosen method to predict the classes for the new iris data:

```
> svmfit = svm(nums~., data=irisdat, kernel="linear", cost = 0.01)
> predict(svmfit, newiris)
 1  2  3  4  5  6  7  8  9 10
 1  1  0  1  1  1  1  0  0  1
Levels: 0 1
# 1 is versicolor, 0 is virginica
```

Now try a radial kernel:

```
> tune.out = tune(svm, Species~., data=irisdat, kernel="radial",
+ ranges=list(cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100), gamma=c(0.5, 1,2, 3, 4)))

> summary(tune.out)

Parameter tuning of svm:

- sampling method: 10-fold cross validation

- best parameters:
```

```
 cost gamma
   10     4

- best performance: 0.02

- Detailed performance results:
    cost gamma error dispersion
1  1e-03   0.5  0.61 0.22827858
2  1e-02   0.5  0.61 0.22827858
3  1e-01   0.5  0.06 0.06992059
4  1e+00   0.5  0.06 0.06992059
5  5e+00   0.5  0.06 0.06992059
6  1e+01   0.5  0.05 0.07071068
7  1e+02   0.5  0.08 0.06324555
8  1e-03   1.0  0.62 0.19888579
9  1e-02   1.0  0.62 0.19888579
10 1e-01   1.0  0.06 0.06992059
11 1e+00   1.0  0.06 0.06992059
12 5e+00   1.0  0.06 0.06992059
13 1e+01   1.0  0.06 0.06992059
14 1e+02   1.0  0.03 0.04830459
15 1e-03   2.0  0.62 0.19888579
16 1e-02   2.0  0.62 0.19888579
17 1e-01   2.0  0.07 0.06749486
18 1e+00   2.0  0.06 0.06992059
19 5e+00   2.0  0.06 0.06992059
20 1e+01   2.0  0.06 0.06992059
21 1e+02   2.0  0.04 0.05163978
22 1e-03   3.0  0.62 0.19888579
23 1e-02   3.0  0.62 0.19888579
24 1e-01   3.0  0.07 0.06749486
25 1e+00   3.0  0.06 0.06992059
26 5e+00   3.0  0.06 0.06992059
27 1e+01   3.0  0.03 0.04830459
28 1e+02   3.0  0.03 0.04830459
29 1e-03   4.0  0.62 0.19888579
30 1e-02   4.0  0.62 0.19888579
31 1e-01   4.0  0.07 0.06749486
32 1e+00   4.0  0.06 0.06992059
33 5e+00   4.0  0.05 0.05270463
34 1e+01   4.0  0.02 0.04216370
35 1e+02   4.0  0.03 0.04830459
```

The best model has a cost of 10 and gamma=4. Its estimated misclassification rate is 0.02. However, costs of 1, 5, 10, and 100 with any value of gamma from 1 to 4 produce results

that do not differ significantly from one another.

Now we'll use the chosen method to predict the classes for the new iris data:

```
svmfit = svm(nums~., data=irisdat, kernel="linear", cost = 0.01)
> predict(svmfit, newiris)
 1  2  3  4  5  6  7  8  9 10
 1  1  0  1  1  0  1  0  0  0
Levels: 0 1
# 1 is versicolor, 0 is virginica
```

Now let's try a random data set.

```
> xr = runif(100, 0, 1)  #Random x-values
> yr = runif(100, 0, 1)  #Random y-values
> crand = sample(1:100, 50, replace=FALSE) #Choose 50 points to be in class 1
> class = rep(-1, 100)
> class[crand] = 1
> srand=cbind(xr, yr, class)
> datrand = data.frame(y=as.factor(srand[,3]), x = srand[,1:2])

> tune.out = tune(svm, y~., data=datrand, kernel="radial",
+ ranges=list(cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100), gamma = c(0.5, 1,2,3,4)))

> summary(tune.out)

Parameter tuning of svm:

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
    1   0.5

- best performance: 0.496

- Detailed performance results:
    cost gamma error dispersion
1  1e-03   0.5 0.540 0.03197221
2  1e-02   0.5 0.545 0.03240370
3  1e-01   0.5 0.501 0.05646041
4  1e+00   0.5 0.496 0.06636599
5  5e+00   0.5 0.496 0.06636599
6  1e+01   0.5 0.496 0.06636599
7  1e+02   0.5 0.496 0.06636599
```

```
8  1e-03   1.0 0.540 0.03197221
9  1e-02   1.0 0.545 0.03240370
10 1e-01   1.0 0.501 0.05646041
11 1e+00   1.0 0.496 0.06636599
12 5e+00   1.0 0.496 0.06636599
13 1e+01   1.0 0.496 0.06636599
14 1e+02   1.0 0.496 0.06636599
15 1e-03   2.0 0.540 0.03197221
16 1e-02   2.0 0.545 0.03240370
17 1e-01   2.0 0.501 0.05646041
18 1e+00   2.0 0.496 0.06636599
19 5e+00   2.0 0.496 0.06636599
20 1e+01   2.0 0.496 0.06636599
21 1e+02   2.0 0.496 0.06636599
22 1e-03   3.0 0.540 0.03197221
23 1e-02   3.0 0.545 0.03240370
24 1e-01   3.0 0.501 0.05646041
25 1e+00   3.0 0.496 0.06636599
26 5e+00   3.0 0.496 0.06636599
27 1e+01   3.0 0.496 0.06636599
28 1e+02   3.0 0.496 0.06636599
29 1e-03   4.0 0.540 0.03197221
30 1e-02   4.0 0.545 0.03240370
31 1e-01   4.0 0.501 0.05646041
32 1e+00   4.0 0.496 0.06636599
33 5e+00   4.0 0.496 0.06636599
34 1e+01   4.0 0.496 0.06636599
35 1e+02   4.0 0.496 0.06636599
```

Many values of the paramters are tied for the best performance of 0.496. Of course the true misclassification rate is 0.5 for any set of parameters. Cross-validation estimates the true misclassification rate well.

9 Clustering

Clustering refers to a class of methods that explore ways to group items. Clustering differs from classification in that there is no training set, nor any a priori specification as to the nature of the groups. Because classification methods are based on information outside of the items to be grouped (e.g. a training set), these methods are referred to as *supervised learning*. Because clustering methods do not use any information outside of the items to be grouped, they are referred to as *unsupervised learning*.

The idea behind clustering methods is that similar items should belong to the same cluster. To implement this idea, there must be a measure of similarity between each pair of items. Such a measure is known as a *similarity score*. The larger the similarity score, the more similar two items are. Alternatively, one may define a measure of distance between any two items. The smaller the distance, the more similar the items are. We will focus on this approach. We begin by defining the notion of distance, or more properly, of a metric.

Definition 9.1. Let S be a nonempty set. A metric on S is a function $d(x, y)$ mapping pairs of items in S to non-negative real numbers, such that

1. For all $x, y \in S$, $d(x, y) = 0$ if and only if $x = y$
2. For all $x, y \in S$, $d(x, y) = d(y, x)$
3. (Triangle Inequality:) For all $x, y, z \in S$, $d(x, y) \leq d(x, z) + d(y, z)$.

Let $\mathbf{x} = (x_1, ..., x_n)^T$ and $\mathbf{y} = (y_1, ..., y_n)^T$. Here are some examples of metrics.

- Euclidean distance: $d(\mathbf{x}, \mathbf{y}) = (\sum (x_i - y_i)^2)^{1/2}$
- Manhattan distance: $d(\mathbf{x}, \mathbf{y}) = \sum |x_i - y_i|$
- Minkowski distance: $d(\mathbf{x}, \mathbf{y}) = (\sum (x_i - y_i)^m)^{1/m}$ Note that Manhattan and Euclidean distances are special cases of Minkowski distance, with $m = 1$ and $m = 2$, respectively.
- Chebychev Distance (Chess distance) $d(x, y) = \max |x_i - y_i|$.
- Identity distance: $d(\mathbf{x}, \mathbf{y}) = 0$ if $\mathbf{x} = \mathbf{y}$, $d(\mathbf{x}, \mathbf{y}) = 1$ if $\mathbf{x} \neq \mathbf{y}$.

9.1 Hierarchical Clustering

Clustering methods are of several types. Hierarchical clustering methods proceed either by successively merging smaller clusters into larger ones (agglomerative hierarchical clustering) or by dividing larger clusters in to smaller ones (divisive hierarchical clustering). Agglomerative methods begin with each item forming its own cluster. The method proceeds in a step-by-step fashion; at each step the most similar clusters are merged until all items are in one cluster. Divisive methods begin with all items in one cluster; at each step, a cluster is divided in two in a way that the items in one of the new clusters are far from the items in the other. This process continues until each item is in its own cluster. We will focus on agglomerative hierarchical clustering.

To decide when to merge or divide clusters, a measure of distance between clusters is needed. Such measures are called *linkage* measures. Some commonly used linkage measures are:

- Single linkage: The distance between clusters is the minimum distance between two points chosen one from each cluster.
- Complete linkage: The distance between clusters is the maximum distance between two points chosen one from each cluster.
- Average linkage: The distance between clusters is the average distance between all pairs of points chosen one from each cluster.
- Centroid distance: The distance between the centroids of the clusters.

Once distances have been defined between all pairs of clusters, the agglomerative hierarchical clustering can begin. Here are the steps.

1. Find the two clusters closest together and merge them into a single cluster (at first each item is its own cluster).
2. Repeat step 1 until all items are in one cluster.

Example 9.1. Following is a distance matrix for five items. Agglomerate using complete linkage.

	1	2	3	4	5
1	0				
2	9	0			
3	3	7	0		
4	6	5	9	0	
5	11	10	2	8	0

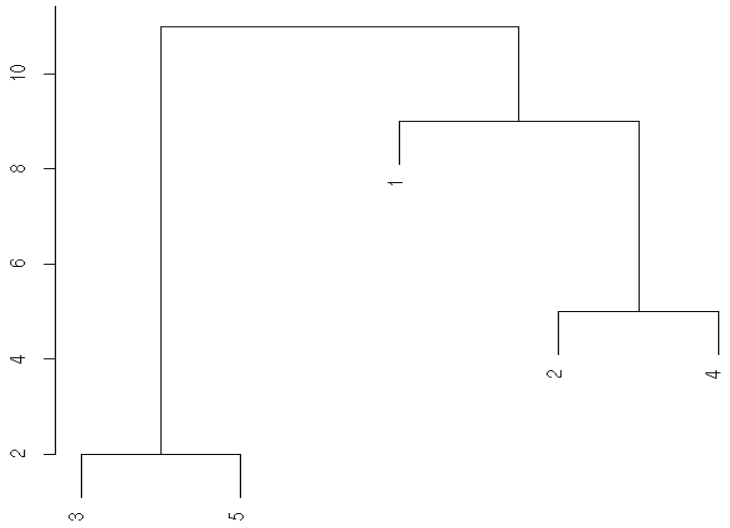
The closest points are 3 and 5. We merge them. Following is the new distance matrix.

	(35)	1	2	4
(35)	0			
1	11	0		
2	10	9	0	
4	9	6	5	0

Now 2 and 4 are most similar. We merge to obtain the next distance matrix.

	(35)	(24)	1
(35)	0		
(24)	10	0	
1	11	9	0

We merge (24) and 1. This leaves two clusters: (35) and (124). Their distance is 11 (the distance between 1 and 5). Following is the dendrogram.



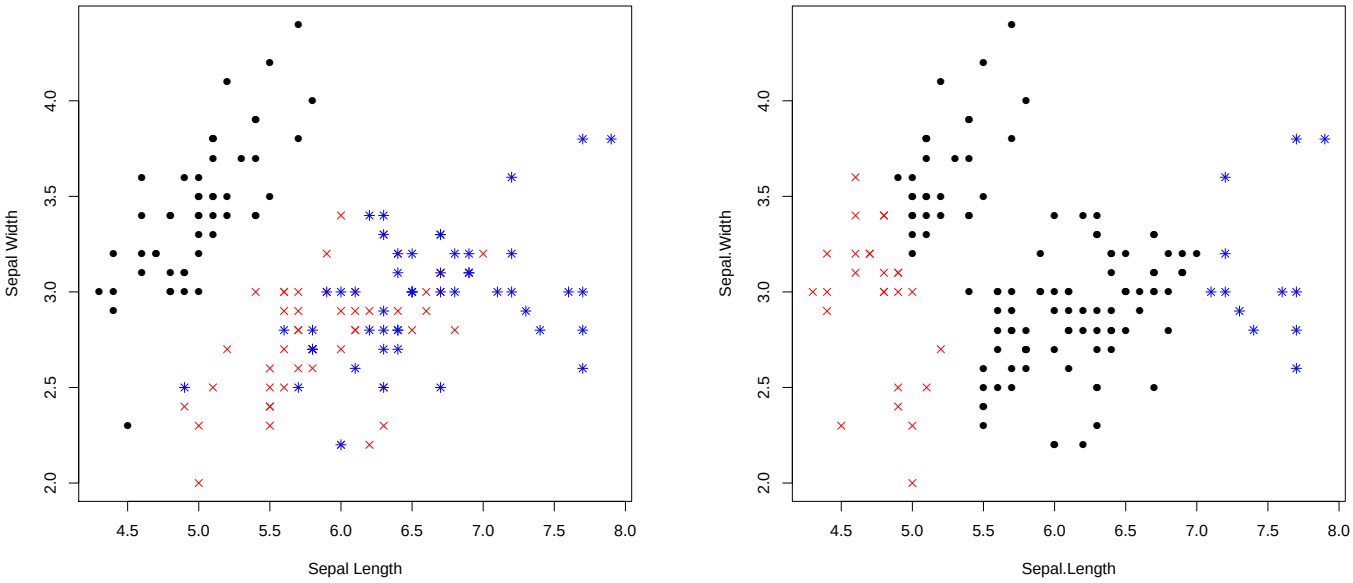
Cluster methods are meant to be exploratory, with the purpose of discovering groups rather than assigning items to predetermined groups. However, we will cluster the iris data to see what happens.

Example 9.2. Cluster the iris data, using agglomerative hierarchical clustering with complete linkage. First just use the sepal length and sepal width.

Solution: We use R.

```
> diris = dist(iris[,1:2])  #Compute matrix of Euclidean distances

#Now cluster. We specify complete linkage, although it is the default
> irisclust = hclust(diris, method = "complete")

Call:
hclust(d = diris)

Cluster method   : complete
Distance         : euclidean
Number of objects: 150

#We need to specify the number of clusters we want. We'll say 3.
> c3 = cutree(irisclust, 3)
```

Following are plots of the true classes (left) and the clusters (right). The clusters don't match the species classes well. Of course, clustering methods are not designed to match previously determined classes. It is conceivable that the clusters have a useful interpretation other than to distinguish the species.

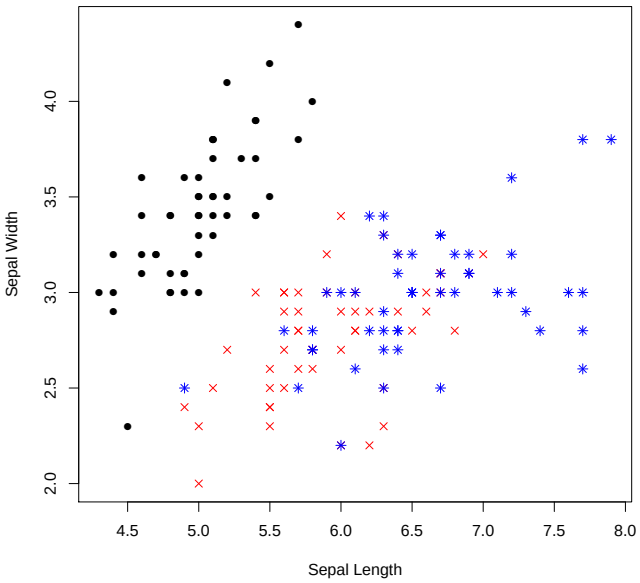


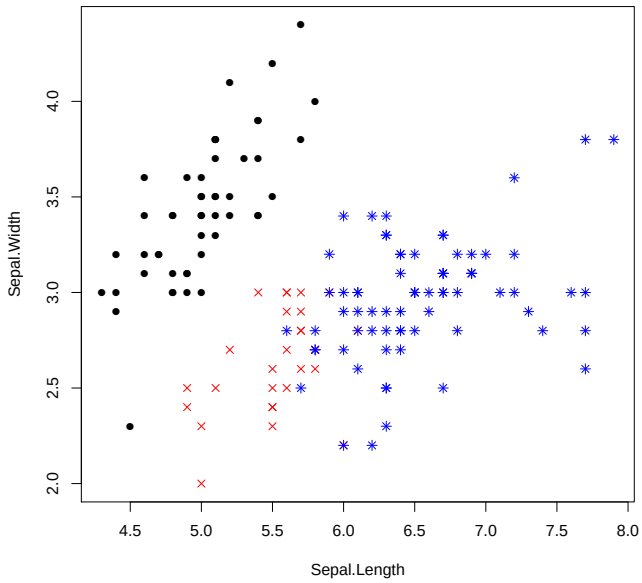
Example 9.3. Classify the iris data, using sepal and petal data.

Solution: We use R.

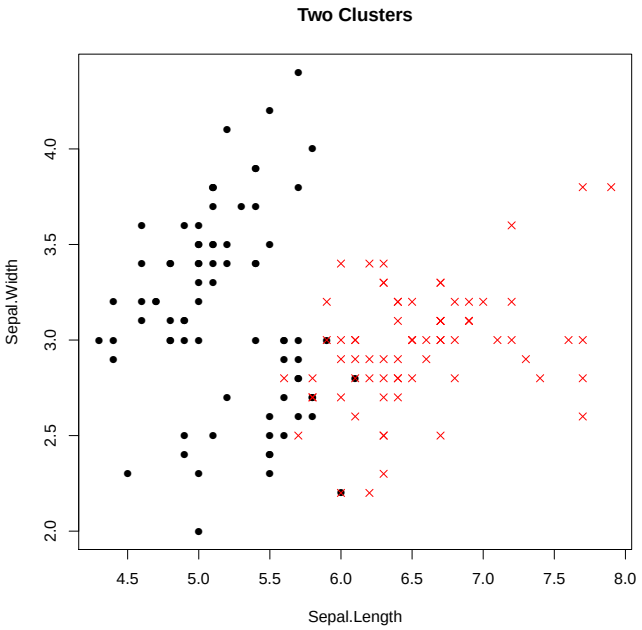
```
> diris = dist(iris[,1:4])  #Compute matrix of Euclidean distances

#Now cluster. We specify complete linkage, although it is the default
> irisclust = hclust(diris, method = "complete")

Call:
hclust(d = diris)

Cluster method   : complete
Distance         : euclidean
Number of objects: 150

#We need to specify the number of clusters we want. We'll say 3.
> c3 = cutree(irisclust, 3)
```

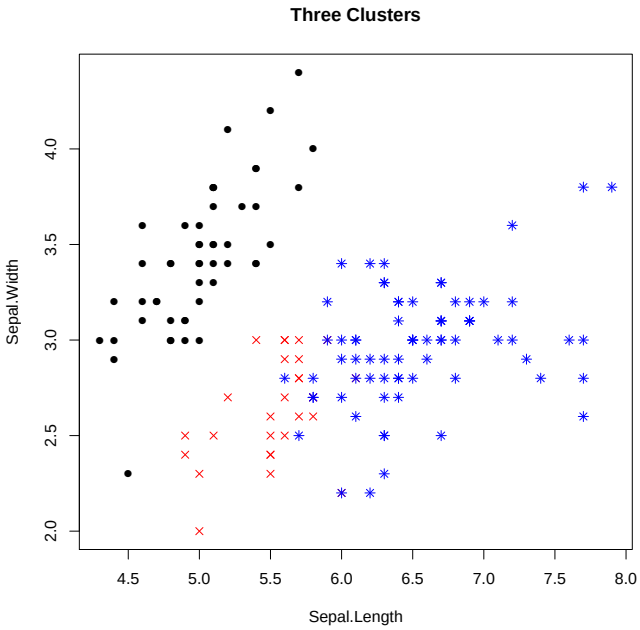
Following are plots of the true classes (left) and the clusters (right). The clusters match the species classes fairly well now.

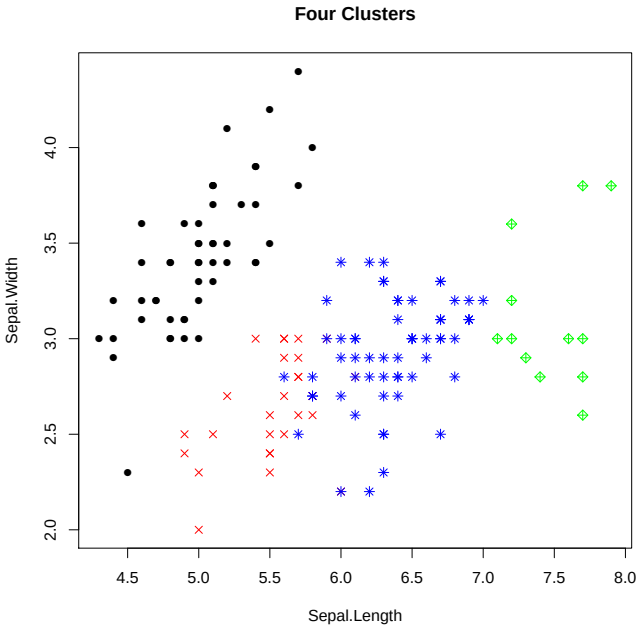


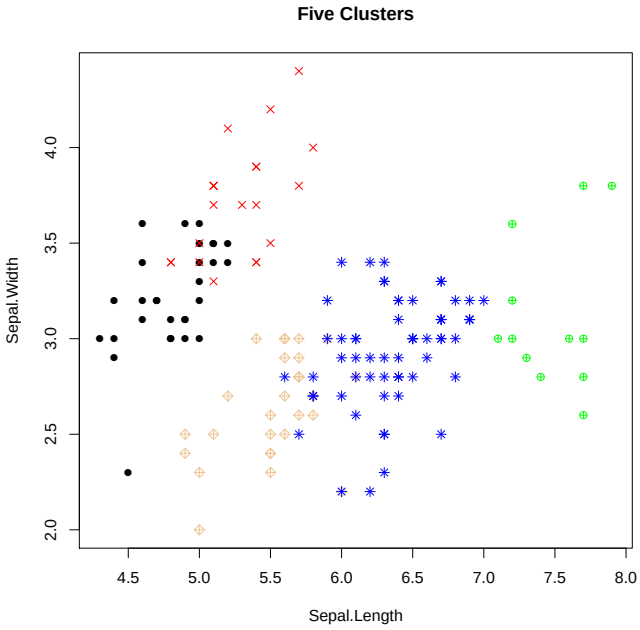


In the preceding examples, we assumed we knew there were three clusters. Following are plots for two through five clusters. How many clusters would you divide these points into?









Choosing the number of clusters

Many methods have been proposed for choosing the number of clusters. Most of them are based on the within groups sum of squares, which we now define. Let k be the number of clusters, let $\mathbf{x}_{i1}, ..., \mathbf{x}_{i,n_i}$ be the items in the ith cluster, and let $\bar{\mathbf{x}}_i$ be the mean of these points, that is, the centroid of the cluster. The within-groups sum of squares is

$$\sum_{i=1}^{k} \sum_{j=1}^{n_i} ||\mathbf{x}_{ij} - \bar{\mathbf{x}}_i||^2$$

In other words, for each point in each cluster, compute the squared distance to the centroid of the cluster, and sum.

One of the methods most often used to choose a number of clusters, and one of the easiest to use, is the *scree plot*. A scree plot is simply a plot of the within-groups sum of squares for various numbers of clusters against the number of clusters. As the number of clusters increases, the within-groups sum of squares will necessarily decrease. One looks for a point at which the decrease slows down noticeably, and chooses that as the number of clusters.

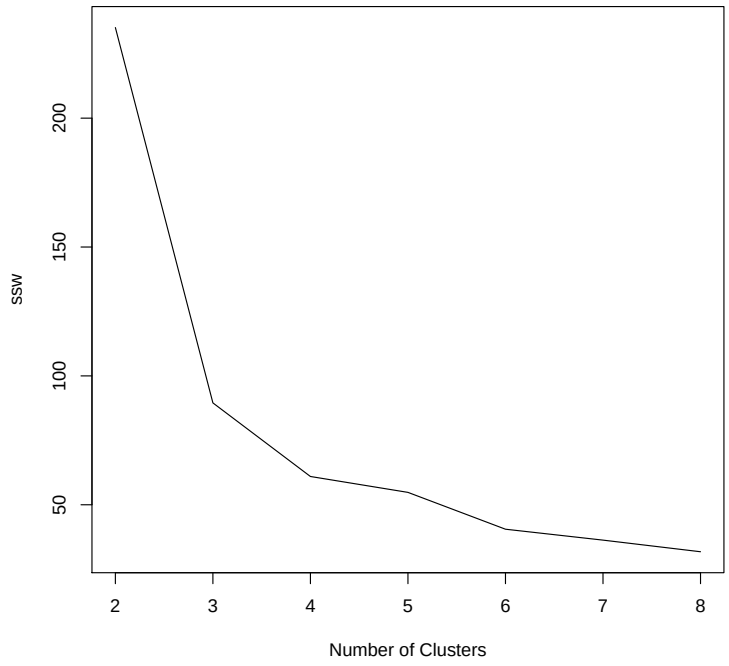
Example 9.4. Using the iris data, compute the within groups sum of squares for hierarchical clustering for two through eight clusters. Construct a scree plot. What number of clusters seems best?

Solution: We use R.

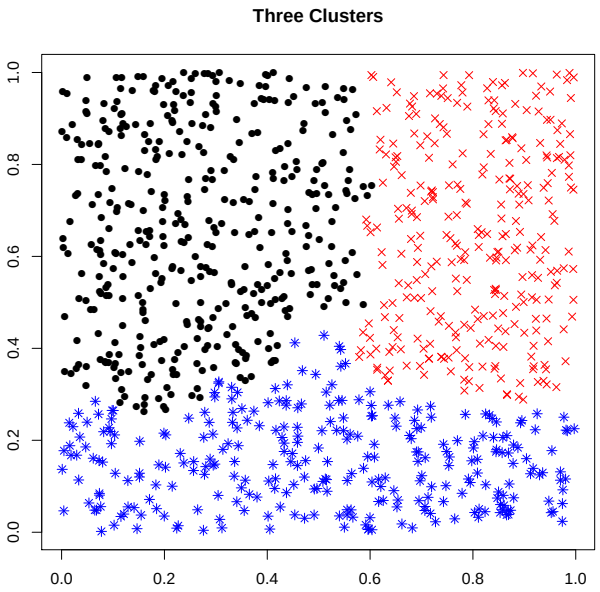
```
> irisdata = iris[,1:4]
> diris = dist(irisdata)
> irisclust = hclust(diris, method = "complete")

#Construct classification vectors for two through eight classes
> c2 = cutree(irisclust, 2)
> c3 = cutree(irisclust, 3)
> c4 = cutree(irisclust, 4)
> c5 = cutree(irisclust, 5)
> c6 = cutree(irisclust, 6)
> c7 = cutree(irisclust, 7)
> c8 = cutree(irisclust, 8)

> cind = cbind(c2, c3, c4, c5, c6, c7, c8)
> ssw = rep(0, 7);

> for (nclust in 2:8) {     #Number of clusters
```

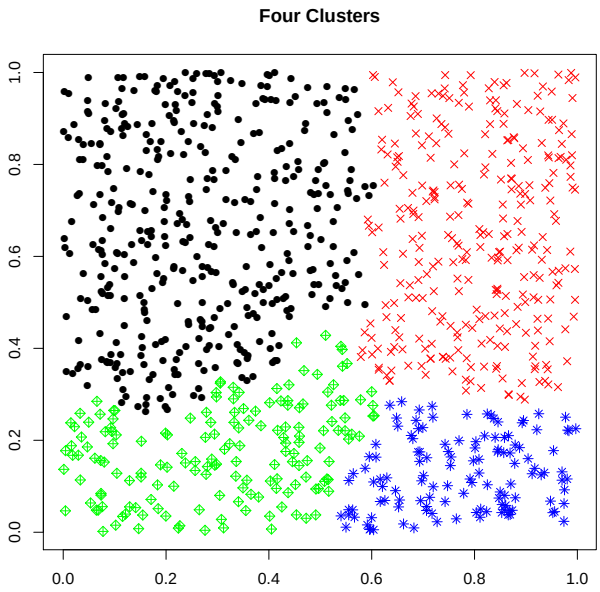
```
+  for (i in 1:nclust) {    #Compute SSW for each cluster and add
+   cdata = irisdata[cind[,nclust-1]==i,]    #Points in the cluster

    #Add sum of squares for this cluster to total
+   ssw[nclust-1] = ssw[nclust-1] + (length(cdata[,1])-1)*sum(diag(var(cdata)))
+  }
+ }

#Plot
> plot(c(2:8), ssw, type="l", xlab="Number of Clusters")
```

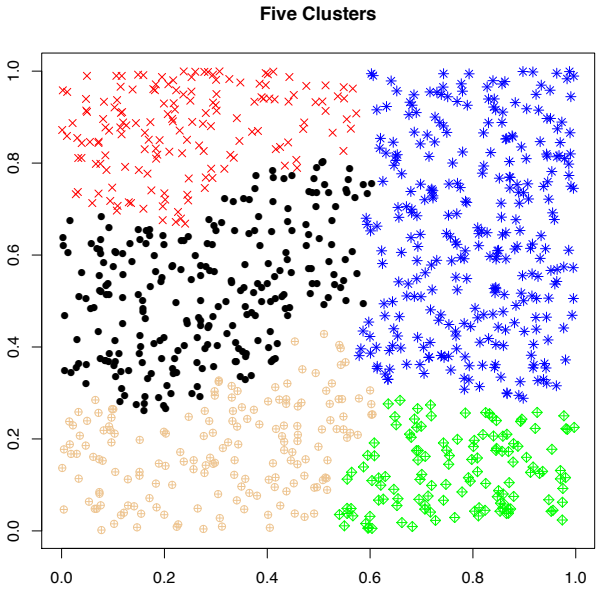
Here is the scree plot. It appears to suggest either four or six clusters.

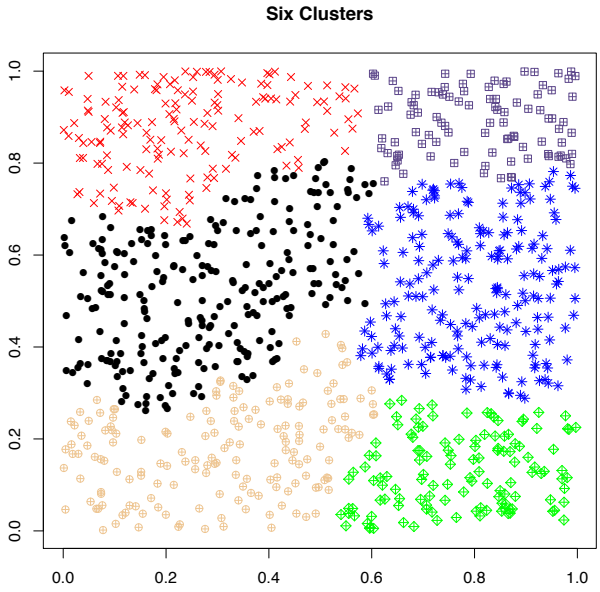


Following are plots of 1000 points randomly generated from the uniform distribution on the unit square. They have been clustered into three, four, five, and six clusters. Can you tell that there are really 1000 clusters?

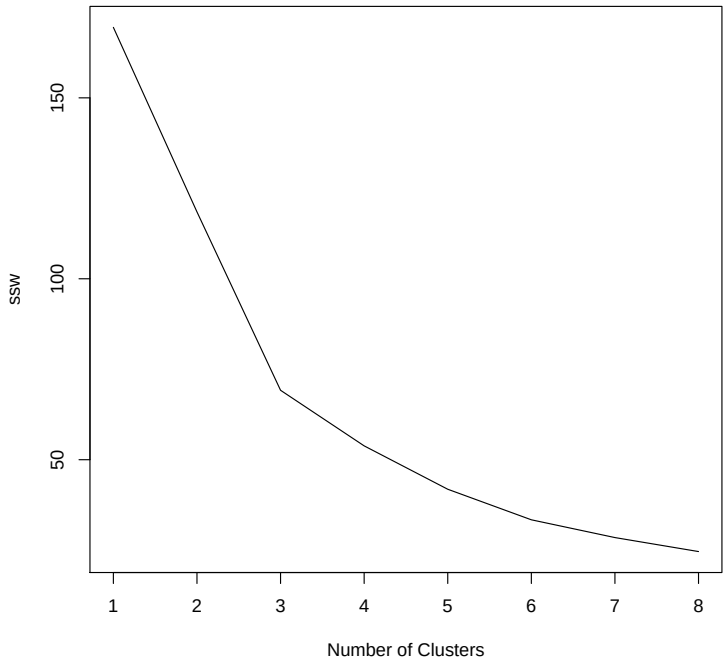




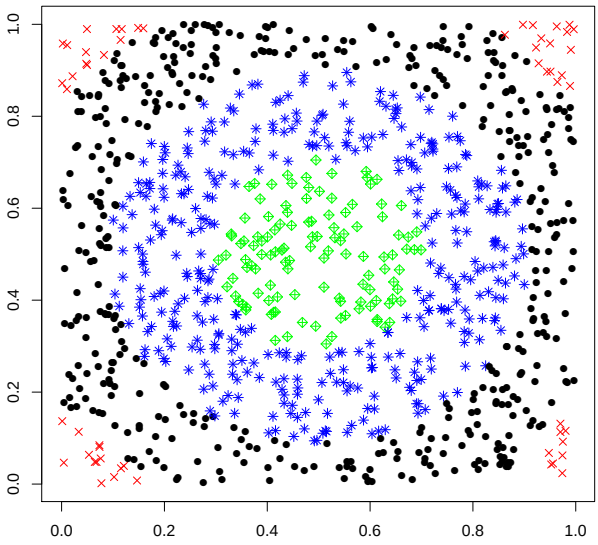


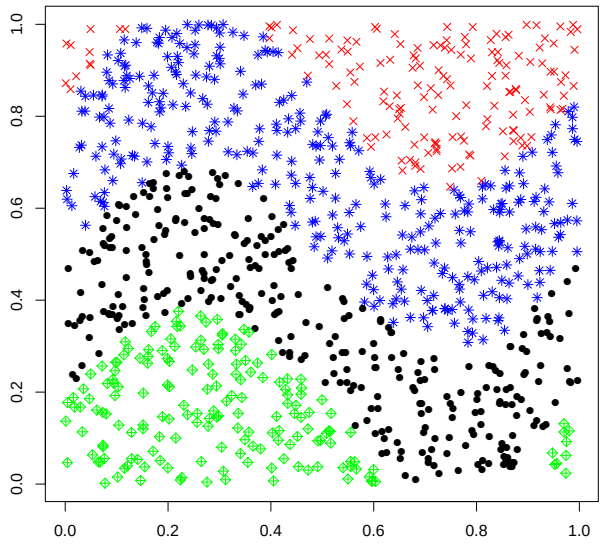


Here is the scree plot for 1 through 8 clusters. There doesn't appear to be a point at which the slope flattens noticeably.



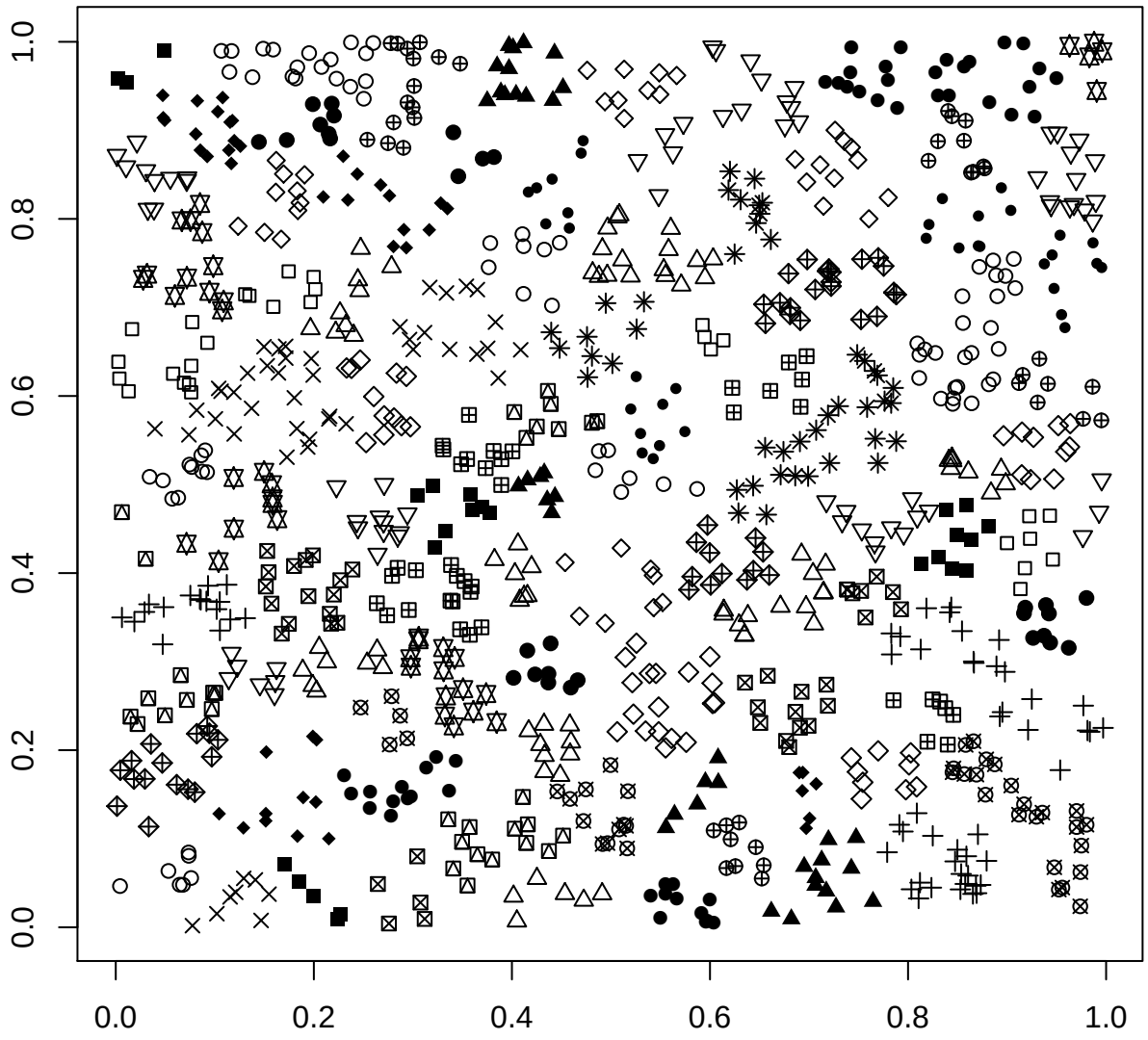
Finally, we explore some alternative metrics to Euclidean distance. In the first, each point is identified with the circle with center at $(1/2, 1/2)$ on which it lies. The distance between two points is the absolute value of the difference between radii of the circles. Following is a plot of random points divided into four clusters. In the second, each point is mapped to the vertical distance to a sine wave with amplitude 0.2. The distance between two points is the absolute value of the difference between their vertical distances.





The form of the clusters depends strongly on the metric. Again, there are actually 1000 clusters in this plot; each point is its own cluster.

Here is a plot of the same points divided into 100 clusters.



9.2 K-Means Clustering

K-means clustering is one of the most commonly used clustering methods. A positive integer k is chosen, and the points are grouped into k clusters. The average of the points in a cluster is the *centroid* of the cluster. The spread of each cluster can be measured by the sum of squared distances from the centroid of the cluster. Specifically, let $\mathbf{x}_{i1}, ..., \mathbf{x}_{im}$ be the points in the ith cluster and let $\bar{\mathbf{x}}_{i.}$ be the centroid. The sum of squares for the ith cluster is

$$\sum_{j=1}^{m} ||\mathbf{x}_i - \bar{\mathbf{x}}_{i.}||^2$$

The within-groups sum of squares for the k clusters is

$$SSW = \sum_{i=1}^{k} \sum_{j=1}^{m} ||\mathbf{x}_{ij} - \bar{\mathbf{x}}_{i.}||^2$$

The optimal clustering is the one that minimizes the within-groups sum of squares.

Unless the number of points to be clustered is very small, it is impossible in practice to find the optimal clustering. Instead, an iterative procedure is used to converge to what is hoped will be a nearly optimal clustering. This method is as follows:

1. Choose k points as initial centroids. They can, for example, be chosen at random.
2. Initially assign each point to be clustered to the centroid closest to it. This creates the initial set of clusters.
3. Compute the centroids of the clusters. They will be different from the initial centroids.
4. For each point, move it to the cluster whose centroid is closest. Recompute the centroids of the clusters after all points are moved.
5. Repeat steps 3 and 4 until no more points are moved.

We present an example.

Example 9.5. Consider the four points in two-dimensional Euclidean space:

A	(5,3)
B	$(-1, 1)$
C	$(1, -2)$
D	$(-3, -2)$

The initial centroids are chosen to be $(0, 1)$ and $(-2, -2)$. Use k-means clustering to group the four points into two clusters.

Solution:
To form the initial clusters, we compute the squared distance from each point to each centroid.

	$(0, 1)$	$(-2, 2)$
A	18	84
B	1	10
C	10	9
D	18	1

The initial clusters are (AB) and (CD). We now compute the centroids of these clusters.

$$C_{AB} = (2, 2) \quad C_{CD} = (-1, -2)$$

We now compute the distances of each point to the new centroids

	$(2, 2)$	$(-1, -2)$
A	10	61
B	10	9
C	17	4
D	41	4

The new clusters are (A) and (BCD). We now compute the centroids of these clusters.

$$C_A = (5, 3) \quad C_{BCD} = (-1, -1)$$

We now compute the distances of each point to the new centroids

	$(2, 2)$	$(-1, -2)$
A	0	52
B	40	4
C	41	5
D	89	5

No points move. The final clusters are (A) and (BCD).

Since there are so few points, we can compute the optimal clustering. There are seven possible clusterings. We compute the within-groups sum of squares for each.

Clustering	SSW
(A), (BCD)	$0 + 14 = 14$
(B), (ACD)	$0 + 48.7 = 48.7$
(C), (ABD)	$0 + 27.7 = 27.7$
(D), (ABC)	$0 + 31.3 = 31.3$
(AB), (CD)	$20 + 8 = 28$
(AC), (BD)	$20.5 + 6.5 = 27$
(AD), (BC)	$44.5 + 6.5 = 51$

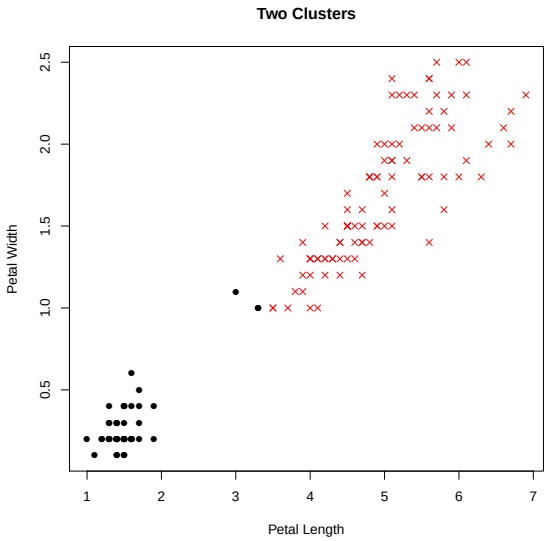
The optimal clustering is (A), (BCD).

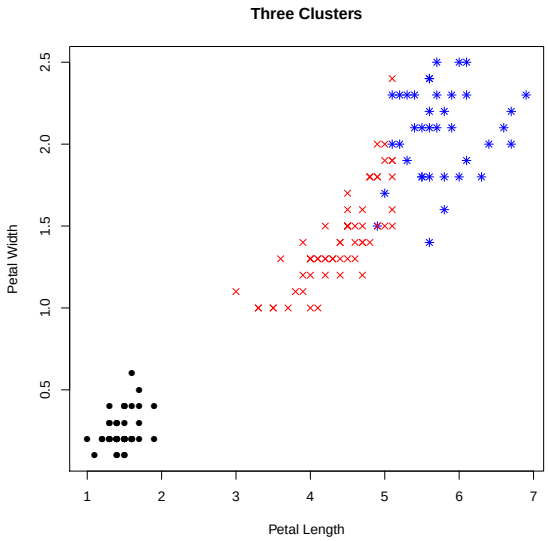
Example 9.6. Use k-means clustering to cluster the iris data. Use $k = 2, 3, 4$, and 5.

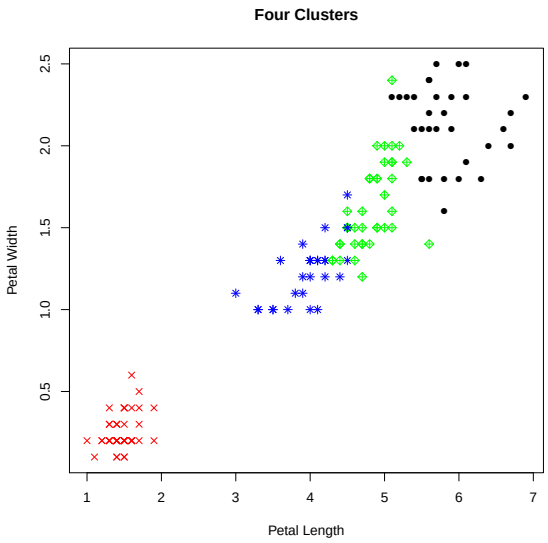
Solution: We use R:

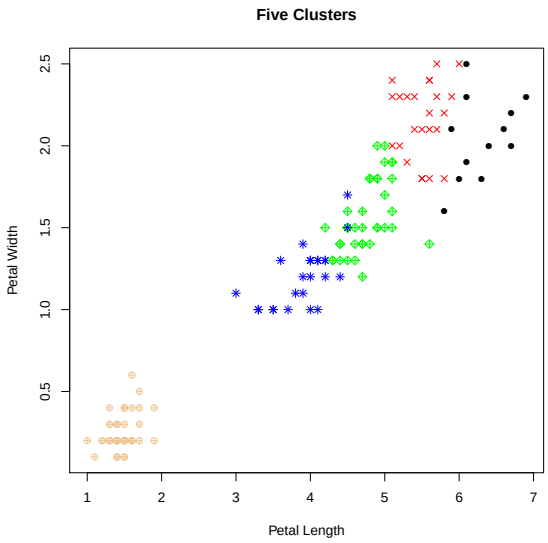
```
> irisdata = iris[,1:4]

> k2 = kmeans(irisdata, 2, nstart=10)  #Compute the clusters
> c2 = k2$cluster                      #Indicator of cluster
> ssw2 = sum(k2$withinss)              #Within groups sum of squares

> k3 = kmeans(irisdata, 3, nstart=10)
> c3 = k3$cluster
> ssw3 = sum(k3$withinss)

> k4 = kmeans(irisdata, 4, nstart=10)
> c4 = k4$cluster
> ssw4 = sum(k4$withinss)

> k5 = kmeans(irisdata, 5, nstart=10)
> c5 = k5$cluster
> ssw5 = sum(k5$withinss)
```

Following are plots for two through five clusters.







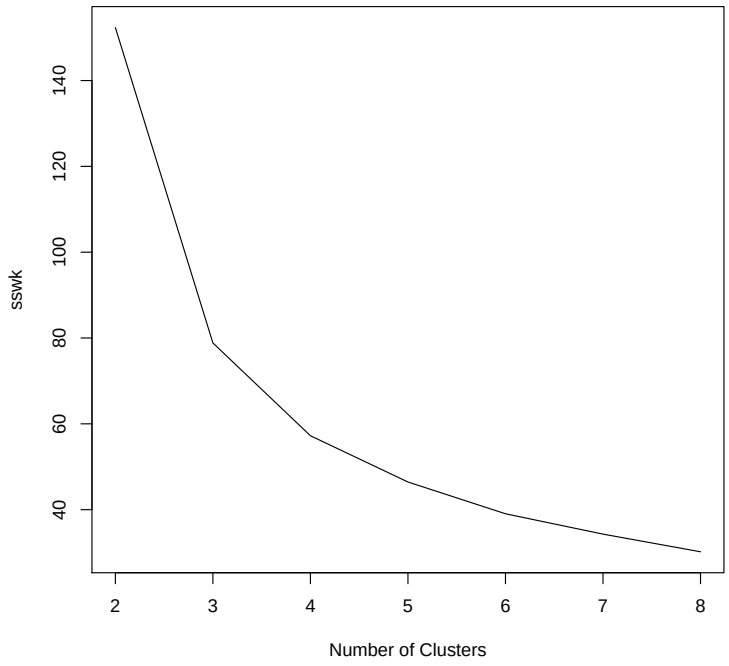


Example 9.7. Using the iris data, compute the within groups sum of squares for k-means clustering for two through eight clusters. Construct a scree plot. What number of clusters seems best?

Solution: We use R.

```
#k2, k3, k4, k5, k6, k7 and k8 are outputs of k-means clustering for k = 2
#through 8.

> ssw = rep(0,7)      #Initialize array of sums of squares

#Within groups sum of squares
> ssw[1] = sum(k2$withinss)
> ssw[2] = sum(k3$withinss)
> ssw[3] = sum(k4$withinss)
> ssw[4] = sum(k5$withinss)
> ssw[5] = sum(k6$withinss)
> ssw[6] = sum(k7$withinss)
> ssw[7] = sum(k8$withinss)

> plot(c(2:8), ssw, type="l". xlab="Number of Clusters")  #Scree plot
```

Here is the scree plot.



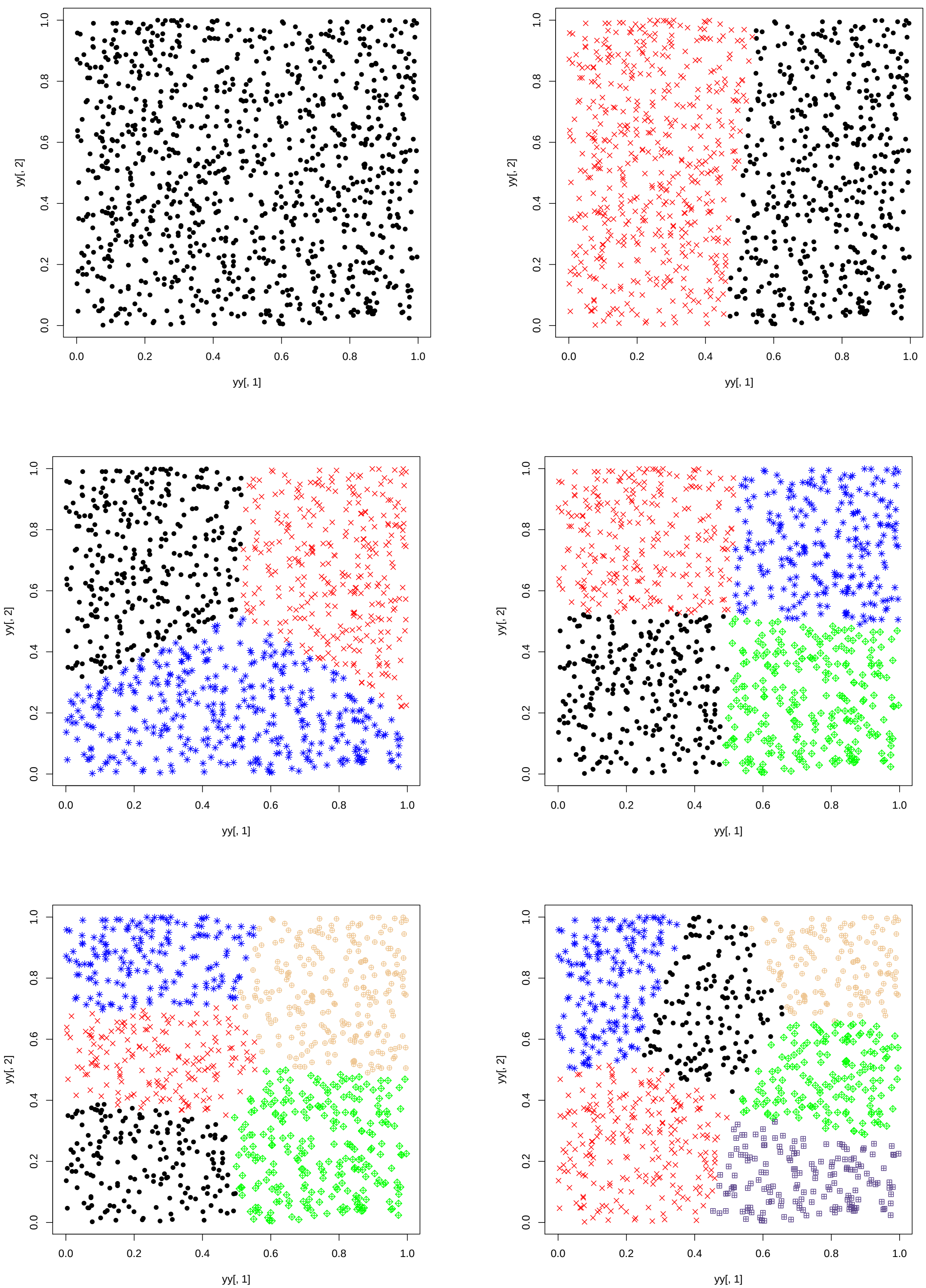
There is no point at which the decrease in slope is particularly dramatic. One would probably choose either three or four clusters.

Example 9.8. Use k-means clustering to cluster a set of 1000 points randomly generated on the unit square. Use $k = 1$ through 6 clusters. Construct a scree plot to determine the number of clusters to choose.

Solution: We use R:

```
> k1 = kmeans(yy, 1, nstart=10)
> c1 = k1$cluster

> k2 = kmeans(yy, 2, nstart=10)
> c2 = k2$cluster

> k3 = kmeans(yy, 3, nstart=10)
> c3 = k3$cluster

> k4 = kmeans(yy, 4, nstart=10)
> c4 = k4$cluster

> k5 = kmeans(yy, 5, nstart=10)
> c5 = k5$cluster

> k6 = kmeans(yy, 6, nstart=10)
> c6 = k6$cluster

> ssw = rep(0, 6)

> ssw[1] = sum(k1$withinss)
> ssw[2] = sum(k2$withinss)
> ssw[3] = sum(k3$withinss)
> ssw[4] = sum(k4$withinss)
> ssw[5] = sum(k5$withinss)
> ssw[6] = sum(k6$withinss)

> plot(c(1:6), ssw, type="l", xlab="Number of Clusters")
```

Following are plots for one through six clusters.



Here is the scree plot. It suggests using four clusters. Of course, there are no meaningful clusters in these data.