REGRESSION ANALYSIS AND PREDICTIVE MODELING -II

DSM-1003

MARCH 21, 2022

MOHAMMAD WASIQ
MS (Data Science

Regression Analysis and Predictive Modeling -II
DSM-1003

Table of Contents

Book :-: **An Introduction to Statistical Learning with Applications in R** by **Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani**
Teacher :-: **Prof. Ahmedur Rehman Sir**
Composer :-: **Mohammad Wasiq** *(Data Science)*

1 Classification

One of the assumptions of linear regression is that the relationship between variables is linear and when the outcome variable is categorical, this assumption is violated. One way around this problem is to transform the data using the logarithmic transformation .

Binary Logistic Regression : When we are trying to predict membership of only two categorical outcomes the analysis is known as binary logistic regression .

1.1 Logistic Regression

Logistic Regression : Logistic regression is multiple regression but with an outcome variable that is a categorical variable and predictor variables that are continuous or categorical.

Logistic Regression Model :

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

Interpretation : The interpretation of logistic regression is the value of the **odds ratio**, which is the exponential of β (i.e., e^{β} or $exp(\beta)$) and is an indicator of the change in odds resulting from a unit change in the predictor .

Task : Now we are fitting the *Logistic Regression* for *Default* data .

Model : $p(default) = \frac{e^{\beta_0+\beta_1(balance)}}{1+e^{\beta_0+\beta_1(balance)}}$

```
library(ISLR2)
data("Default")
head(Default)

##   default student   balance    income
## 1      No      No  729.5265 44361.625
## 2      No     Yes  817.1804 12106.135
## 3      No      No 1073.5492 31767.139
## 4      No      No  529.2506 35704.494
## 5      No      No  785.6559 38463.496
## 6      No     Yes  919.5885  7491.559

# Fitting the Logistic Model :

log_m <- glm(default ~ balance , data = Default , family = binomial)
log_m

##
## Call:  glm(formula = default ~ balance, family = binomial, data = Default)
##
## Coefficients:
## (Intercept)      balance
```

```
##  -10.651331     0.005499
##
## Degrees of Freedom: 9999 Total (i.e. Null);  9998 Residual
## Null Deviance:       2921
## Residual Deviance: 1596  AIC: 1600
```

```
# Summary of Model
summary(log_m)
```

```
##
## Call:
## glm(formula = default ~ balance, family = binomial, data = Default)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2697  -0.1465  -0.0589  -0.0221   3.7589
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.065e+01  3.612e-01  -29.49   <2e-16 ***
## balance      5.499e-03  2.204e-04   24.95   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1596.5  on 9998  degrees of freedom
## AIC: 1600.5
##
## Number of Fisher Scoring iterations: 8
```

Fitted Model : $\hat{p}(default) = \frac{e^{-10.65+0.0055(balance)}}{1+e^{-10.65+0.0055(balance)}}$

Interpretation : A one-unit increase in **balance** is associated with an increase in the log odds of **default** by **0.0055** units. **OR** $\hat{p}(default) = \frac{e^{-10.65+0.0055\times1000}}{1+e^{-10.65+0.0055\times1000}} = 0.00576$ 1000 unit increase in **balance** is associated with an increase in the log odds of **default** by **5** units with probability **0.00576** .

Model : $p(default) = \frac{e^{\beta_0+\beta_1(student)}}{1+e^{\beta_0+\beta_1(student)}}$

```
# Fitting the Logistic Model :

log_s <- glm(default ~ student , data = Default , family = binomial)
# Summary of Model
summary(log_s)
```

```
##
## Call:
```

```
## glm(formula = default ~ student, family = binomial, data = Default)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.2970  -0.2970  -0.2434  -0.2434   2.6585
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.50413    0.07071  -49.55  < 2e-16 ***
## studentYes   0.40489    0.11502    3.52 0.000431 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 2908.7  on 9998  degrees of freedom
## AIC: 2912.7
##
## Number of Fisher Scoring iterations: 6
```

Model : $p(default) = \frac{e^{-3.50+0.4049(student)}}{1+e^{-3.50+0.4049(student)}}$

Interpretation : A one-unit increase in **student** is associated with an increase in the log odds of **default** by **0.4049** units.

$$\widehat{Pr}(default = Yes|student = Yes) = \frac{e^{-3.50+0.4049\times 1}}{1 + e^{-3.50+0.4049\times 1}} = 0.0431$$

$$\widehat{Pr}(default = Yes|student = Yes) = \frac{e^{-3.50+0.4049\times 0}}{1 + e^{-3.50+0.4049\times 0}} = 0.0292$$

1.2 Multiple Logistic Regression

$$p(X) = \frac{e^{\beta_0+\beta_1 X_1+\ldots+\beta_p X_p}}{1 + e^{\beta_0+\beta_1 X_1+\ldots+\beta_p X_p}}$$

Model : $p(default) = \frac{e^{\beta_0+\beta_1(balance)+\beta_2(income)+\beta_3(student)}}{1+e^{\beta_0+\beta_1(balance)+\beta_2(income)+\beta_3(student)}}$

```
# Fitting the Logistic Model :

log_bis <- glm(default ~ balance + income + student , data = Default , family
= binomial)
# Summary of Model
summary(log_bis)

##
## Call:
## glm(formula = default ~ balance + income + student, family = binomial,
```

```
##     data = Default)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4691  -0.1418  -0.0557  -0.0203   3.7383
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.087e+01  4.923e-01 -22.080  < 2e-16 ***
## balance      5.737e-03  2.319e-04  24.738  < 2e-16 ***
## income       3.033e-06  8.203e-06   0.370  0.71152
## studentYes  -6.468e-01  2.363e-01  -2.738  0.00619 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1571.5  on 9996  degrees of freedom
## AIC: 1579.5
##
## Number of Fisher Scoring iterations: 8
```

Fitted Model : $p(default) = \frac{e^{-10.869+0.0057(balance)+0.0030(income)-0.6468(student)}}{1+e^{-10.869+0.0057(balance)+0.0030(income)-0.6468(student)}}$

The negative coefficient for *student* in the multiple logistic regression indicates that for a fixed value of *balance* and *income* , a student is less likely to default than a non-student.

A student with a credit card balance of *$1500* and an income of *$40000* has an estimated probability of default of $\hat{p}(X) = \frac{e^{-10.869+0.0057\times1500+0.0030\times40-0.6468\times1}}{1+e^{-10.869+0.0057\times1500+0.0030\times40-0.6468\times1}} = 0.058$

A non-student with the same balance and income has an estimated probability of default of $\hat{p}(X) = \frac{e^{-10.869+0.0057\times1500+0.0030\times40-0.6468\times0}}{1+e^{-10.869+0.0057\times1500+0.0030\times40-0.6468\times0}} = 0.105$

we multiply the income coefficient estimate by *40*, rather than by 40,000, because in that table the model was fit with income measured in units of *$1, 000*)

1.3 Multinomial Logistic Regression

Multinomial / Polychotomous Logistic Regression : When we want to predict membership of more than two categories we use Multinomial (or Polychotomous) Logistic Regression. It turns out that it is possible to extend the two-class logistic regression approach to the setting of *K > 2* classes.

$$Pr(Y = K|X = x) = \frac{e^{\beta_{k_0}+\beta_{k_1}x_1+\ldots+\beta_{k_p}x_p}}{1 + \sum_{i=1}^{k-1} e^{\beta_{k_0}+\beta_{k_1}x_1+\ldots+\beta_{k_p}x_p}}$$

for $k = 1, \ldots, k\text{-}1$ and $Pr(Y = K|X = x) = \frac{1}{1+\sum_{i=1}^{k-1} e^{\beta_{k_0}+\beta_{k_1}x_1+\ldots+\beta_{k_p}x_p}}$

It is not hard to show that for $k = 1,\ldots,K-1$,

$$log(\frac{Pr(Y = k|X = x)}{Pr(Y = K|X = x)}) = \beta_{k0} + \beta_{k1}x_1 + \cdots + \beta_{kp}x_p$$

The decision to treat the *Kth* class as the baseline is unimportant . The coefficient estimates will differ between the two fitted models due to the differing choice of baseline, but the fitted values (predictions), the log odds between any pair of classes, and the other key model outputs will remain the same.

We now briefly present an alternative coding for multinomial logistic regression, known as the **softmax coding**. The softmax coding is equivalent softmax to the coding just described in the sense that the fitted values, log odds between any pair of classes, and other key model outputs will remain the same, regardless of coding. In the softmax coding, rather than selecting a baseline class, we treat all K classes symmetrically, and assume that for $k = 1,\ldots,K$.

$$Pr(Y = k|X = x) = \frac{e^{\beta_{k0}+\beta_{k_1}x_1+\ldots+\beta_{k_p}x_p}}{\sum_{i=1}^{K} e^{\beta_{l_0}+\beta_{l_1}x_1+\ldots+\beta_{l_p}x_p}}$$

Thus, rather than estimating coefficients for $K-1$ classes, we actually estimate coefficients for all K classes. The log odds ratio between the kth and $k'th$ classes equals . It is not hard to show that for $k = 1,\ldots,K-1$,

$$log(\frac{Pr(Y = k|X = x)}{Pr(Y = k'|X = x)}) = (\beta_{k0} - \beta_{k'0}) + (\beta_{k1} - \beta_{k'1})x_1 + \cdots + (\beta_{kp} - \beta_{k'p})x_p$$

1.4 Generative Models for Classification

Logistic regression involves directly modeling $Pr(Y = k|X = x)$ using the logistic function .

Why do we need another method, when we have logistic regression ?
There are several reasons: - When there is substantial separation between the two classes, the parameter estimates for the logistic regression model are surprisingly unstable. The methods that we consider in this section do not suffer from this problem. - If the distribution of the predictors X is approximately normal in each of the classes and the sample size is small, then the approaches in this section may be more accurate than logistic regression. - The methods in this section can be naturally extended to the case of more than two response classes. (In the case of more than two response classes, we can also use multinomial logistic regression)

Suppose that we wish to classify an observation into one of K classes, where $K \geq 2$. In other words, the qualitative response variable Y can take on K possible distinct and unordered values . Let π_k represent the *overall* or *prior probability* that a randomly chosen observation comes from the prior *kth* class. Let $f_k(X) \equiv Pr(X|Y = k)$ denote the density function of X for an observation that comes from the *kth* class. In other words, $f_k(x)$ is relatively large if there is a high probability that an observation in the kth class has $X \approx x$, and $f_k(x)$ is small if it is very unlikely that an observation in the *kth* class has $X \approx x$.

Then *Bayes' theorem* states that :

$$Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{k} \pi_l \, f_l(x)}$$

we will use the abbreviation $p_k(x) = Pr(Y = k|X = x)$; this is the posterior probability that an observation posterior $X = x$ belongs to the *kth* class. That is, it is the probability that the observation belongs to the *kth* class, given the predictor value for that observation.

we discuss three classifiers that use different estimates of $f_k(x)$ to approximate the Bayes classifier : - **Linear Discriminant Analysis (LDA) - Quadratic Discriminant Analysis (QDA) - Naive Bayes (NB)**

1.4.1 Linear Discriminant Analysis (LDA) for p = 1

Assume that $p = 1$ that is, we have only *one predictor. We will then classify an observation to the class for which* $p_k(x)$ *is greatest. To estimate* $f_k(x)$ *,we will first make some assumptions about its form . In particular, we assume that* $f_k(x)$ *is* normal or *Gaussian*. In the one- normal Gaussian dimensional setting, the normal density takes the form

$$f_k(x) = \frac{1}{\sigma_k\sqrt{2\pi}} exp\left(-\frac{1}{2\sigma_k^2}(x-\mu_k)^2\right)$$

where, μ_k ans σ_k^2 are the *mean* and *variance* parameters for the *kth* class. For now, let us further assume that $\sigma_1^2 = \sigma_2^2 = \cdots = \sigma_k^2$: that is, there is a shared variance term across all K classes, which for simplicity we can denote by σ^2 .

$$p_k(x) = \frac{\pi_k \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{1}{2\sigma^2}(x-\mu_k)^2\right)}{\sum_{l=1}^{K} \pi_l \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{1}{2\sigma^2}(x-\mu_l)^2\right)}$$

π_k denotes the *prior probability* that an observation belongs to the *kth* class.

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

For instance, if $K = 2$ and $\pi_1 = \pi_2$, then the Bayes classifier assigns an observation to class 1 if $2x(\mu_1 - \mu_2) > \mu_1^2 - \mu_2^2$, and to class 2 otherwise. The Bayes decision boundary is the point for which $\delta_1(x) = \delta_2(x)$;one can show that this amounts to $x = \frac{\mu_1+\mu_2}{2}$

In practice, even if we are quite certain of our assumption that X is drawn from a Gaussian distribution within each class, to apply the Bayes classifier we still have to estimate the parameters $\mu_1, \ldots., \mu_2$, $\pi_1, \ldots, \pi_2$ and σ^2. The *Linear Discriminant Analysis (LDA)* method approximates the *Bayes classifier* by plugging estimates for π_k, μ_k, σ^2 In particular, the following estimates are used :

$$\widehat{\mu_k} = \frac{1}{n_k}\sum_{i-1}^{k} x_i$$

$$\widehat{\sigma^2} = \frac{1}{n-K}\sum_{k=1}^{K}\sum_{i:y_i=k}(x_i - \widehat{\mu_k})^2$$

where n is the total number of training observations and n_k is the number of training observations in the *kth* class. LDA estimates π_k using the proportion of the training observations that belongs to the *kth* class. In other words, $\widehat{\pi_k} = n_k/n$. The LDA classifier plugs the estimates and assigns an observation *X = x* to the class for which

$$\hat{\delta}_k(x) = x.\frac{\hat{\mu}_k}{\hat{\sigma}^2} - \frac{\hat{\mu}_k^2}{2\hat{\sigma}^2} + log(\hat{\pi}_k)$$

is largest. The word linear in the classifier's name stems from the fact that the discriminant functions $\hat{\delta}_k(x)$ are linear functions of x (as opposed to a more complex function of x).

1.4.2 Linear Discriminant Analysis for p > 1

We now extend the LDA classifier to the case of multiple predictors. To do this, we will assume that $X = (X_1, X_2, \ldots, X_p)$ is drawn from a multi-variate *Gaussian* (or *multivariate normal*) distribution, with a class-specific multivariate mean vector and a common covariance matrix.

To indicate that a p-dimensional random variable X has a multi-variate Gaussian distribution, we write **X ~ N(μ, Σ)**. Here **E(X) = μ** is the mean of X (a vector with p components), and **Cov(X) = Σ** is the *p × p* covariance matrix of X. Formally, the *multivariate Gaussian density* is defined as

$$f(x) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} exp[-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)]$$

In the case of $p > 1$ predictors, the LDA classifier assumes that the observations in the *kth* class are drawn from a *multivariate Gaussian distribution* $N(\mu_k, \Sigma)$, where μ_k is a class-specific mean vector, and $\sum$ is a covariance matrix that is common to all K classes. Plugging the density function for the *kth* class, $f_k(X = x)$, and performing a little bit of algebra reveals that the *Bayes classifier* assigns an observation $X = x$ to the class for which

$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + log\,\pi_k$$

is largest . Three equally-sized Gaussian classes are shown with class-specific mean vectors and a common covariance matrix. The three ellipses represent regions that contain *95 %* of the probability for each of the three classes. The dashed lines are the Bayes decision boundaries. In other words, they represent the set of values x for which $\delta_k(x) = \delta_l(x)$, i.e

$$x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k = x^T\Sigma^{-1}\mu_l - \frac{1}{2}\mu_l^T\Sigma^{-1}\mu_l$$

for $k \neq l$.

1.4.3 Quadratic Descriminant Analysis (QDA)

As we have discussed, LDA assumes that the observations within each class are drawn from a multivariate Gaussian distribution with a class-specific mean vector and a covariance matrix that is common to all K classes.

Quadratic discriminant analysis (QDA) provides an alternative approach. Like LDA, the QDA classifier results from assuming that the observations from each class are drawn from a Gaussian distribution, and plugging estimates for the parameters into Bayes' theorem in order to perform prediction. However, unlike LDA, QDA assumes that each class has its own covariance matrix. That is, it assumes that an observation from the kth class is of the form $X \sim N(\mu_k, \Sigma_k)$, where Σ_k is a covariance matrix for the *kth* class. Under this assumption, the Bayes classifier assigns an observation *X = x* to the class for which

$$\delta_k(x) = -\frac{1}{2}(x-\mu_k)^T\Sigma^{-1}\mu_k(x-\mu_k) - \frac{1}{2}log|\Sigma_k| + log\,\pi_k$$

$$= -\frac{1}{2}x^T\Sigma_k^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma_k^{-1}\mu_k - \frac{1}{2}log|\Sigma_k| + log\,\pi_k$$

is largest.

Why would one prefer *LDA* to *QDA*, or vice-versa ?
The answer lies in the bias-variance trade-off. When there are *p* predictors, then estimating a covariance matrix requires estimating $p(p+1)/2$ parameters. *QDA* estimates a separate covariance matrix for each class, for a total of $K_p(p+1)/2$ parameters. By instead assuming that the *K classes* share a common covariance matrix, the *LDA* model becomes *linear in x*, which means there are K_p linear coefficients to estimate. Consequently, **LDA is a much less flexible classifier than QDA, and so has substantially lower variance .** Roughly speaking, **LDA tends to be a better bet than QDA if there are relatively few training observations and so reducing variance is crucial**. In contrast, *QDA is recommended if the training set is very large*, so that the variance of the classifier is not a major concern, or if the assumption of a common covariance matrix for the *K classes* is clearly untenable.

The QDA decision boundary is inferior, because it suffers from higher variance without a corresponding decrease in bias. In contrast, the right-hand panel displays a situation in which the orange class has a *correlation* of *0.7* between the variables and the blue class has a correlation of *−0.7*. Now the Bayes decision boundary is quadratic, and so QDA more accurately approximates this boundary than does LDA.

1.4.4 Naive Bayes

$f_k(x)$ is the *p-dimensional* density function for an observation in the *kth* class, for *k = 1,…,K*. In general, estimating a p-dimensional density function is challenging. In LDA, we make a very strong assumption that greatly simplifies the task: we assume that fk is the density function for a multivariate normal random variable with class-specific mean μ_k, and shared covariance matrix Σ. By contrast, in QDA, we assume that f_k is the density function for a multivariate normal random variable with class-specific mean μ_k, and class-specific covariance matrix Σ_k. By making these very strong assumptions, we are able to replace the very challenging problem of estimating K p-dimensional density functions with the much simpler problem of estimating K p-dimensional mean vectors and one (in the case of LDA) or K (in the case of QDA) (p x p)-dimensional covariance matrices.

The naive Bayes classifier takes a different tack for estimating $f_1(x), \ldots, f_K(x)$. Instead of assuming that these functions belong to a particular family of distributions
(e.g. multivariate normal), we instead make a single assumption :

Within the kth class, the p predictors are independent. Stated mathematically, this assumption means that for *k = 1,…,K,*

$$f_k(x) = f_{k1}(x_1) \times f_{k2}(x_2) \times \cdots \times f_{kp}(x_p)$$

where f_{kj} is the density function of the jth predictor among observations in the *kth* class.

Why is this assumption so powerful ?
Essentially, estimating a p-dimensional density function is challenging because we must consider not only the *marginal distribution* of each predictor — that is, the distribution of each predictor on its own — but also the joint distribution of the predictors — that is, the association between the different predictors. In the case of a multivariate normal distribution, the association between the different predictors is summarized by the off-diagonal elements of the covariance matrix. However, in general, this association can be very hard to characterize, and exceedingly challenging to estimate. But by assuming that the p covariates are independent within each class, we completely eliminate the need to worry about the association between the p predictors, because we have simply assumed that there is no association between the predictors!

In fact, since estimating a joint distribution requires such a *huge amount of data, naive Bayes is a good choice* in a wide range of settings. Essentially, the naive Bayes assumption introduces some bias, but reduces variance, leading to a classifier that works quite well in practice as a result of the bias-variance trade-off.

$$Pr(Y = k|X = x) = \frac{\pi_k \times f_{k1}(x_1) \times f_{k2}(x_2) \times \cdots \times\times f_{kp}(x_p)}{\sum_{l=1}^{K} \pi_l \times f_{l1}(x_1) \times f_{l2}(x_2) \times \cdots \times\times f_{lp}(x_p)}$$

for $k = 1,2, \ldots, K$.
To estimate the one-dimensional density function f_{kj} using training data $x_{ij}, \ldots, x_{nj}$, we have a few options . - If X_j is quantitative, then we can assume that $X_j|Y = k \sim N(\mu jk \, , \, \sigma_{jk}^2)$
In other words, we assume that within each class, the jth predictor is drawn from a (univariate) normal distribution. While this may sound a bit like QDA, there is one key difference, in that here we are assuming that the predictors are independent; this amounts to QDA with an additional assumption that the class-specific covariance matrix is diagonal.

- If X_j is quantitative, then another option is to use a non-parametric estimate for f_{kj} . A very simple way to do this is by making a histogram for the observations of the jth predictor within each class. Then we can estimate $f_{kj}(x_j)$ as the fraction of the training observations in the kth class that belong to the same histogram bin as x_j . Alternatively, we can use a kernel density estimator, which is essentially a smoothed version of a histogram . For instance, suppose that $X_j \in \{\mathbf{1, 2, 3}\}$, and we have 100 observations in the *kth* class. Suppose that the *jth* predictor takes on values of *1, 2* and *3* in *32, 55* and *13* of those observations, respectively. Then we can estimate f_{kj} as

1.4 Comparision of Classification Methods

1.4.5 An Analytical Comparison

We perform a *analytical* (or mathematical) comparison of *LDA, QDA, Naive Bayes* and *Logistic Regression* . We consider these approaches in a setting with *K* classes, so that we assign an observation to the class that maximizes *Pr(Y=k | X=x)*. Equivalently we can set *K* as the *baseline* class and assign an observation to the class that maximizes

$$log(\frac{Pr(Y = k|X = x)}{Pr(Y = K|X = x)}) \quad ; \quad for \;\; k = 1, \ldots, K$$

First, for *LDA*, we can make use of *Bayes' Theorem* as well as the assumption that the predictors within each class are drawn from a *multivariate normal density* with *class-specific mean* and shared *covariance* matrix in order to show that

$$log(\frac{Pr(Y = k|X = x)}{Pr(Y = K|X = x)}) = a_k + \sum_{j=1}^{p} b_{kj}\; x_j \quad \ldots (i)$$

where, $a_k = log(\frac{\pi_k}{\pi_K}) - \frac{1}{2}(\mu_k + \mu_K)^T \Sigma^{-1}(\mu_k - \mu_K)$ and b_{kj} is the j^{th} component of $\Sigma^{-1}(\mu_k - \mu_K)$. Hence *LDA* , like logistic regression, assumes that the log adds of the *posterior probability* is *linear* in x.

Using similar calculations, in the *QDA* setting become

$$log(\frac{Pr(Y = k|X = x)}{Pr(Y = K|X = x)}) = a_k + \sum_{j=1}^{p} b_{kj}\, x_j + \sum_{j=1}^{p}\sum_{l=1}^{p} c_{kjl}\; x_j\, x_l \quad \ldots (ii)$$

where, a_k, b_{kj} and c_{kjl} are functions of π_k, π_K, μ_k, μ_K, Σ_k and Σ_K. Again as the name suggests, *QDA* assumes that the *log odds* of the *posterior probabilities* is *quadratic* in x .

Finally, we examine in the naive Bayes setting. Recall that in this setting, $f_k(x)$ is modeled as a product of p one-dimensional functions $f_{kj}(x_j)$ for $j = 1, \ldots p$.
Hence,

$$log(\frac{Pr(Y=k|X=x)}{Pr(Y=K|X=x)}) = a_k + \sum_{j=1}^{p} g_{kj}\ x_j \quad \dots(iii)$$

where, $a_k = log(\frac{\pi_k}{\pi_K})$ and $g_{kj}(x_j) = log(\frac{f_{kj}(x_j)}{f_{Kj}(x_j)})$. Hence, the right-hand side of above equation takes the form of a *generalized additive model* .

Inspection of (i), (ii), and (iii) yields the following observations about *LDA, QDA* and *Naive Bayes* :

- *LDA* is a special case of *QDA* with $c_{kjl} = 0$ for all $j = 1, \dots, p\ \ ,\ l = 1, \dots, p\ and\ k = 1, \dots, K$ (Of course, this is not surprising, since LDA is simply a restricted version of QDA with $\Sigma_1 = \cdots = \Sigma_K = \Sigma$)
- Any classifier with a *linear decision boundary* is a special case of *naive Bayes* with $g_{kj}(x_j) = b_{kj}\, x_j$ In particular, this means that *LDA* is a special case of *naive Bayes!* This is not at all obvious from the descriptions of *LDA and naive Bayes* earlier in the chapter, since each method makes very different assumptions: *LDA* assumes that the features are *normally distributed* with a common within-class *covariance matrix*, and *naive Bayes* instead assumes *independence of the features*
- If we model $f_{kj}(x_j)$ in the *naive Bayes* classifier using a one-dimensional *Gaussian distribution* $N(\mu_{kj}, \sigma_j^2)$, then we end up with $g_{kj}(x_j) = b_{kj}\, x_j$ where $b_{kj} = (\mu_{kj} - \mu_{Kj})/\sigma_j^2$. In this case, *naive Bayes* is actually a special case of *LDA* with Σ restricted to be a diagonal matrix with j^{th} diagonal element equal to σ_j^2.
- Neither *QDA* nor *Naive Bayes* is a special case of the other. *Naive Bayes* can produce a more flexible fit, since any choice can be made for $g_{kj}(x_j)$. However, it is restricted to a purely additive fit, in the sense that in (iii), a function of x_j is added to a function of x_l, for $j \neq l$; however, these terms are never multiplied. By contrast, QDA includes multiplicative terms of the form $c_{kjl}\, x_j\, x_l$. Therefore, QDA has the potential to be more accurate in settings where interactions among the predictors are important in discriminating between classes.

None of these methods uniformly dominates the others: in any setting, the choice of method will depend on the true distribution of the predictors in each of the *K classes*, as well as other considerations, such as the values of *n* and *p*. The latter ties into the bias-variance trade-off. From Logistic Regression :

$$log(\frac{Pr(Y=k|X=x)}{Pr(Y=K|X=x)}) = \beta_{k0} + \sum_{j=1}^{p} \beta_{kj}\ x_j$$

This is identical to the *linear form of LDA* (ii): in both cases, $log(\frac{Pr(Y=k|X=x)}{Pr(Y=K|X=x)})$ is a linear function of the predictors. In *LDA*, the coefficients in this linear function to functions of estimates for π_k, π_K, μ_k, μ_K and Σ obtained by assuming that $X_1, \ldots, X_p$ follows a *normal distribution* within each class. By contrast, in *logistic regression*, the coefficients are chosen to maximize the likelihood function. Thus, we expect *LDA* to outperform *logistic regression* when the *normality* assumption (appxrox.) holds, and we expect logistic regression to perform better when it does not.

In order to make a prediction for an observation $X = x$, the training observations that are closest to x are identified. Then X is assigned to the class to which the plurality of these observations belong. Hence KNN is a completely non-parametric approach : no assumptions are made about the shape of the decision boundary. We make the following observations about KNN :

- Because **KNN** is completely non-parametric, we can expect this approach to boundary is *highly non-linear*, provided that n is *very* large and p is *small*.
- In order to provide accurate classification, **KNN** requires a *lot* of observations relative to the number of predictors—that is, n much *larger* than p. This has to do with the fact that *KNN* is *non-parametric*, and thus tends to reduce the bias while incurring a lot of variance.
- In settings where the decision boundary is non-linear but n is only *modest* or p is *not very small*, then **QDA** may be *preferred* to **KNN**. This is because *QDA* can provide a *non-linear decision boundary* while taking advantage of a parametric form, which means that it requires a smaller sample size for accurate classification, relative to KNN.
- Unlike logistic regression, *KNN* does *not tell* us which *predictors* are important, so we don't get a table of coefficients.

1.4.6 An Empirical Comparison

we compare the *empirical* (practical) performance of *Logistic Regression, LDA, QDA, Naive Bayes* and *KNN*. We generated data from six different *scenarios*, each of which involves a binary (two-class) classification problem. In *three* of the scenarios, the Bayes decision *boundary is linear*, and in the *remaining* scenarios it is *non-linear*.

Scenario are on **ISLR**'s Page **162/607**

1.5 Generalized Linear Models

We assumed that *response* Y is *quantitative* and *explored* the use of *least squares linear regression* to predict Y. However, we may sometimes be faced with situations in which Y is neither *quantitative* nor *qualitative* and so neither *linear regresion* .

1.5.1 Linear Regression on the Bikeshare Data

1.5.2 Poisson Regression on the Bikeshare Data

1.5.3 Generalized Linear Models in Greater Generality

We have now discussed three types of regression models : *linear, logistic and Poisson*. These approaches share some common characteristics :

1. Each approach uses *predictors* $X_1, \ldots, X_p$ to predict a *response* Y . We assume that, conditional on $X_1, \ldots, X_p$, Y belongs to a certain family of distributions. For *Linear Regression*, we typically assume that Y follows a *Gaussian* or *Normal Distribution*. i.e $Y \overset{LM}{\sim} N(\mu, \sigma^2)$ For *Logistic Regression*, we assume that Y follows a *Bernoulli Distribution*. i.e. $Y \overset{Log}{\sim} Bernoulli\big(p,\ (1-p)\big)$ For *Poisson Regression* we assume that **Y* follows a *Poisson Distribution*. i.e. $Y \overset{Pois}{\sim} Poisson(\lambda)$

2. Each approach models the mean of Y as a function of the predictors. In *Linear Regression*, the mean of Y takes the form

$$E\big(Y|X_1, \ldots, X_p\big) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p \quad \cdots (i)$$

i.e. it is a linear function of the predictors. For *Logistic Regression*, the mean instead takes the form

$$E\big(Y|X_1, \ldots, X_p\big) = Pr\big(Y = 1|X_1, \ldots, X_p\big)$$
$$= \frac{e^{\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p}}{1 + e^{\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p}} \quad \cdots (ii)$$

For *Poisson Regression* it takes the form

$$E\big(Y|X_1, \ldots, X_p\big) = \lambda\big(X_1, \ldots, X_p\big) = e^{\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p} \quad \cdots (iii)$$

Equations *(i)–(iii)* can be expressed using a *link function*, η, which link function applies a transformation to $E\big(Y|X_1, \ldots, X_p\big)$ so that the transformed mean is a linear function of the predictors. That is,

$$\eta\left(E\big(Y|X_1, \ldots, X_p\big)\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$$

The link functions for **Linear, Logistic** and **Poisson Regression** are $\eta(\mu) = \mu$, $\eta(\mu) = log(\frac{\mu}{(1-\mu)})$ and $\eta(\mu) = log(\mu)$, respectively .

The *Gaussian, Bernoulli* and *Poisson Distributions* are all members of a wider class of distributions, known as the *exponential family*. Other well-known members of this family are the *exponential distribution*, the *Gamma Distribution*, and the *Negative Binomial Distribution*. In general, we can perform a *regression* by modeling the *response* Y as coming from a particular member of the *exponential* family and then transforming the mean of the response so that the transformed mean is a linear function of the predictors via (iii). **Any**

regression approach that follows this very general recipe is known as a Generalized Linear Model(GLM). Thus, *Linear Regression, Logistic Regression* and *Poisson Regression* are three examples of *GLMs* . Other examples not covered here include *Gamma regression* and *negative binomial regression* .

1.6 Lab : Classification Methods

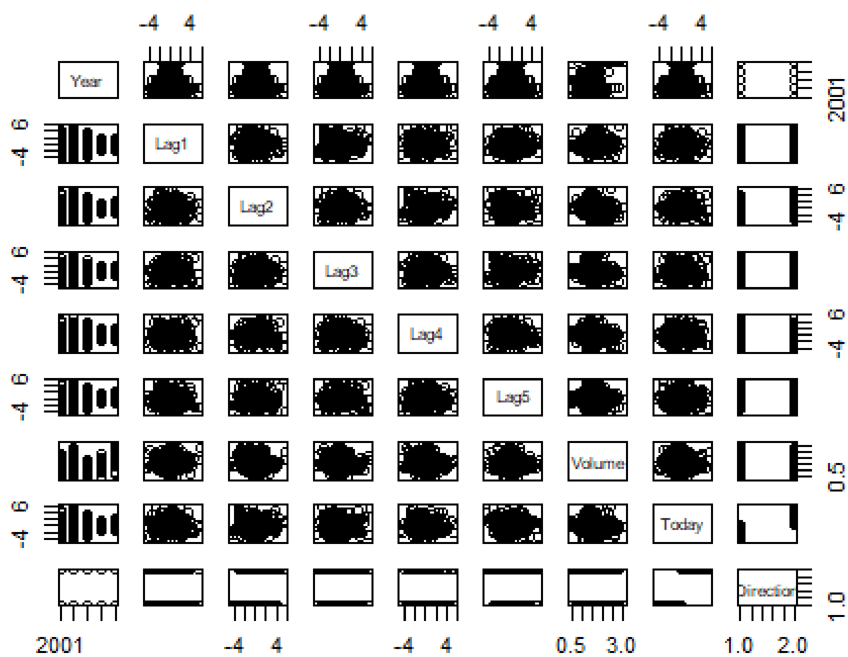
1.6.0.1 The Stock Market Data

We will begin by examining some numerical and graphical summaries of the *Smarket* data, which is part of the *ISLR2* library. This data set consists of percentage returns for the S&P 500 stock index over 1, 250 days, from the beginning of 2001 until the end of 2005. For each date, we have recorded the percentage returns for each of the five previous trading days, *Lag1* through *Lag5*. We have also recorded *Volume* (the number of shares traded on the previous day, in billions), *Today* (the percentage return on the date in question) and *Direction* (whether the market was Up or Down on this date). Our goal is to predict Direction (a qualitative response) using the other features.

```
# Load the Library
library(ISLR2)

# load Data
data("Smarket")

# names of columns
names(Smarket)

## [1] "Year"      "Lag1"      "Lag2"      "Lag3"      "Lag4"
## [6] "Lag5"      "Volume"    "Today"     "Direction"

# Dimension of Data
dim(Smarket)

## [1] 1250    9

# head of data
head(Smarket)

##   Year   Lag1   Lag2   Lag3   Lag4   Lag5 Volume  Today Direction
## 1 2001  0.381 -0.192 -2.624 -1.055  5.010 1.1913  0.959        Up
## 2 2001  0.959  0.381 -0.192 -2.624 -1.055 1.2965  1.032        Up
## 3 2001  1.032  0.959  0.381 -0.192 -2.624 1.4112 -0.623      Down
## 4 2001 -0.623  1.032  0.959  0.381 -0.192 1.2760  0.614        Up
## 5 2001  0.614 -0.623  1.032  0.959  0.381 1.2057  0.213        Up
## 6 2001  0.213  0.614 -0.623  1.032  0.959 1.3491  1.392        Up

# Summary of Data
summary(Smarket)

##       Year           Lag1                Lag2
##  Min.   :2001   Min.   :-4.922000   Min.   :-4.922000
```

```
##  1st Qu.:2002   1st Qu.:-0.639500   1st Qu.:-0.639500
##  Median :2003   Median : 0.039000   Median : 0.039000
##  Mean   :2003   Mean   : 0.003834   Mean   : 0.003919
##  3rd Qu.:2004   3rd Qu.: 0.596750   3rd Qu.: 0.596750
##  Max.   :2005   Max.   : 5.733000   Max.   : 5.733000
##       Lag3                Lag4                Lag5
##  Min.   :-4.922000   Min.   :-4.922000   Min.   :-4.92200
##  1st Qu.:-0.640000   1st Qu.:-0.640000   1st Qu.:-0.64000
##  Median : 0.038500   Median : 0.038500   Median : 0.03850
##  Mean   : 0.001716   Mean   : 0.001636   Mean   : 0.00561
##  3rd Qu.: 0.596750   3rd Qu.: 0.596750   3rd Qu.: 0.59700
##  Max.   : 5.733000   Max.   : 5.733000   Max.   : 5.73300
##      Volume            Today             Direction
##  Min.   :0.3561   Min.   :-4.922000   Down:602
##  1st Qu.:1.2574   1st Qu.:-0.639500   Up  :648
##  Median :1.4229   Median : 0.038500
##  Mean   :1.4783   Mean   : 0.003138
##  3rd Qu.:1.6417   3rd Qu.: 0.596750
##  Max.   :3.1525   Max.   : 5.733000
```

```
# Matrix Plot
pairs(Smarket)
```

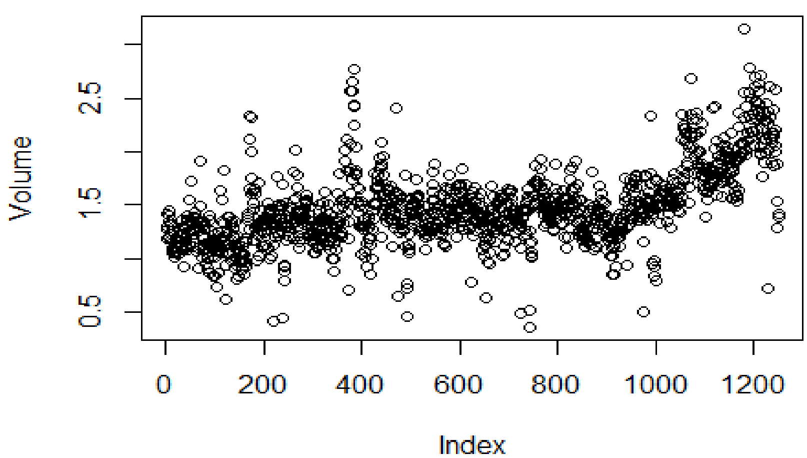



```
# Correlation of Numeric Data
cor(Smarket[, -9]) # 9th Col is not numeric
```

```
##                Year          Lag1          Lag2          Lag3
## Year   1.00000000  0.029699649  0.030596422  0.033194581
## Lag1   0.02969965  1.000000000 -0.026294328 -0.010803402
## Lag2   0.03059642 -0.026294328  1.000000000 -0.025896670
## Lag3   0.03319458 -0.010803402 -0.025896670  1.000000000
## Lag4   0.03568872 -0.002985911 -0.010853533 -0.024051036
## Lag5   0.02978799 -0.005674606 -0.003557949 -0.018808338
## Volume 0.53900647  0.040909908 -0.043383215 -0.041823686
## Today  0.03009523 -0.026155045 -0.010250033 -0.002447647
##                Lag4          Lag5      Volume         Today
## Year    0.035688718  0.029787995  0.53900647  0.030095229
## Lag1   -0.002985911 -0.005674606  0.04090991 -0.026155045
## Lag2   -0.010853533 -0.003557949 -0.04338321 -0.010250033
## Lag3   -0.024051036 -0.018808338 -0.04182369 -0.002447647
## Lag4    1.000000000 -0.027083641 -0.04841425 -0.006899527
## Lag5   -0.027083641  1.000000000 -0.02200231 -0.034860083
## Volume -0.048414246 -0.022002315  1.00000000  0.014591823
## Today  -0.006899527 -0.034860083  0.01459182  1.000000000
```

As one would expect, the correlations between the lag variables and today's returns are close to zero. In other words, there appears to be little correlation between today's returns and previous days' returns. The only substantial correlation is between Year and Volume. By plotting the data, which is ordered chronologically, we see that Volume is increasing over time. In other words, the average number of shares traded daily increased from 2001 to 2005 .

```
# Convert the Data into attach


# Plot The Volume Column
plot(Volume)
```



1.6.1 Logistic Regression

We will fit a logistic regression model in order to predict Direction using *Lag1* through *Lag5* and Volume. The `glm()` function can be used to fit many types of *generalized linear models*, including *logistic regression*. The syntax of the `glm()` function is similar to that of `lm()`, except that we must pass in the argument `family = binomial` in order to tell R to run a logistic regression rather than some other type of generalized linear model

Model :

$$Direction = \frac{e^{\beta_0+\beta_1\cdot Lag1+\beta_2\cdot Lag2+\beta_3\cdot Lag3+\beta_4\cdot Lag4+\beta 5\cdot Lag5+\beta_6\cdot Volume}}{1+e^{\beta_0+\beta_1\cdot Lag1+\beta_2\cdot Lag2+\beta_3\cdot Lag3+\beta_4\cdot Lag4+\beta 5\cdot Lag5+\beta_6\cdot Volume}} + \epsilon$$

```
# Fit the Logistic Regression
log_m <- glm(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume ,
data = Smarket ,
family = binomial)

# Summary of Model
summary(log_m)

##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
##     Volume, family = binomial, data = Smarket)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.446  -1.203   1.065   1.145   1.326
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.126000   0.240736  -0.523    0.601
## Lag1        -0.073074   0.050167  -1.457    0.145
## Lag2        -0.042301   0.050086  -0.845    0.398
## Lag3         0.011085   0.049939   0.222    0.824
## Lag4         0.009359   0.049974   0.187    0.851
## Lag5         0.010313   0.049511   0.208    0.835
## Volume       0.135441   0.158360   0.855    0.392
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1731.2  on 1249  degrees of freedom
## Residual deviance: 1727.6  on 1243  degrees of freedom
## AIC: 1741.6
##
## Number of Fisher Scoring iterations: 3
```

Fitted Model :

$$\widehat{Direction} = \frac{e^{-0.126-0.073\cdot Lag1-0.042\cdot Lag2+0.011\cdot Lag3+0.009\cdot Lag4+0.010\cdot Lag5+0.0135\cdot Volume}}{1+e^{-0.126-0.073\cdot Lag1-0.042\cdot Lag2+0.011\cdot Lag3+0.009\cdot Lag4+0.010\cdot Lag5+0.0135\cdot Volume}}$$

The smallest p-value here is associated with *Lag1*. The negative coefficient for this predictor suggests that if the market had a positive return yesterday, then it is less likely to go up today. However, at a value of *0.15*, the p-value is still relatively large, and so there is no clear evidence of a real association between *Lag1 and Direction*.

Coefficient :

```
# Coefficient
coef(log_m)

##  (Intercept)         Lag1         Lag2         Lag3         Lag4
## -0.126000257 -0.073073746 -0.042301344  0.011085108  0.009358938
##         Lag5       Volume
##  0.010313068  0.135440659

# Summary of Coefficient
summary(log_m)$coef

##                 Estimate Std. Error    z value  Pr(>|z|)
## (Intercept) -0.126000257 0.24073574 -0.5233966 0.6006983
## Lag1        -0.073073746 0.05016739 -1.4565986 0.1452272
## Lag2        -0.042301344 0.05008605 -0.8445733 0.3983491
## Lag3         0.011085108 0.04993854  0.2219750 0.8243333
## Lag4         0.009358938 0.04997413  0.1872757 0.8514445
## Lag5         0.010313068 0.04951146  0.2082966 0.8349974
## Volume       0.135440659 0.15835970  0.8552723 0.3924004

summary(log_m)$coef[, 4]

## (Intercept)        Lag1        Lag2        Lag3        Lag4
##   0.6006983   0.1452272   0.3983491   0.8243333   0.8514445
##        Lag5      Volume
##   0.8349974   0.3924004
```

Predicted Values : - The `predict()` function can be used to predict the probability that the market will go up, given values of the predictors. The `type = "response"` option tells R to output probabilities of the form $P(Y = 1|X)$, as opposed to other information such as the logit. If no data set is supplied to the *predict()* function, then the probabilities are computed for the training data that was used to fit the *logistic regression model* .

We know that these values correspond to the probability of the market going up, rather than down, because the `contrasts()` function indicates that R has created a dummy variable with a 1 for *Up*.

```
# Predicted Values
glm.probs <- predict (log_m , type = "response")
```

```r
# First 10 predicted Values
glm.probs[1:10]
```

```
##         1         2         3         4         5         6
## 0.5070841 0.4814679 0.4811388 0.5152224 0.5107812 0.5069565
##         7         8         9        10
## 0.4926509 0.5092292 0.5176135 0.4888378
```

```r
# Direction Contast
contrasts(Direction)
```

```
##      Up
## Down  0
## Up    1
```

In order to make a prediction as to whether the market will go up or down on a particular day, we must convert these predicted probabilities into class labels, Up or Down. The following two commands create a vector of class predictions based on whether the predicted probability of a market increase is greater than or less than 0.5.

```r
glm.pred <- rep (" Down ", 1250)

glm.pred[glm.probs > 0.5] = "Up"
```

The first command creates a vector of *1,250* `Down` elements. The second line transforms to Up all of the elements for which the predicted probability of a market increase exceeds 0.5. Given these predictions, the `table()` function can be used to produce a *confusion matrix* in order to determine how many observations were correctly or incorrectly classified.

```r
table (glm.pred , Direction)
```

```
##         Direction
## glm.pred Down  Up
##     Down  145 141
##    Up     457 507
```

The *diagonal elements* of *the confusion matrix* indicate *correct predictions*, while the *off-diagonals* represent *incorrect predictions*.

Hence our model correctly predicted that the market would go up on *507* days and that it would go down on *145* days, for a total of 507 + 145 = *652* correct predictions.

The `mean()` function can be used to compute the fraction of days for which the prediction was correct.

```r
(507 + 145) / 1250
```

```
## [1] 0.5216
```

```r
mean(glm.pred == Direction)
```

```
## [1] 0.4056
```

In this case, *logistic regression* correctly predicted the movement of the market 40.5% of the time.

At first glance, it appears that the logistic regression model is working a little better than random guessing.

However, this result is misleading because we trained and tested the model on the same set of *1,250* observations. In other words, $100\% - 40.5\% = 59.5\%$, is the *training* error rate . In order to better assess the accuracy of the logistic regression model in this setting, we can fit the model using part of the data, and then examine how well it predicts the held out data. This will yield a more realistic error rate, in the sense that in practice we will be interested in our model's performance not on the data that we used to fit the model, but rather on days in the future for which the market's movements are unknown.

To implement this strategy, we will first create a vector corresponding to the observations from *2001 through 2004*. We will then use this vector to create a held out data set of observations from *2005*.

```
train <- (Year < 2005)

Smarket.2005 <- Smarket[!train, ]

dim(Smarket.2005)

## [1] 252   9

Direction.2005 <- Direction[!train]
```

The Object *train* is a vector of 1,250 elements, corresponding to the observations in our data set. The elements of the vector that correspond to observations that occurred before 2005 are set to *TRUE*, whereas those that correspond to observations in 2005 are set to *FALSE*. The object train is a *Boolean* vector, since its elements are TRUE and FALSE. For instance, the command *Smarket[train,]* would pick out a *sub-matrix* of the stock market data set, corresponding only to the dates before 2005, since those are the ones for which the elements of train are TRUE. The ! symbol can be used to reverse all of the elements of a Boolean vector.

We now fit a logistic regression model using only the subset of the observations that correspond to dates before 2005, using the subset argument. We then obtain predicted probabilities of the stock market going up for each of the days in our test set—that is, for the days in 2005.

```
glm.fits <- glm (Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume ,
data = Smarket ,
family = binomial ,
subset = train)

glm.probs <- predict (glm.fits , Smarket.2005,
type = "response")
```

Notice that we have trained and tested our model on two completely separate data sets: training was performed using only the dates before 2005 and testing was performed using only the dates in 2005. Finally, we compute the predictions for 2005 and compare them to the actual movements of the market over that time period.

```
glm.pred <- rep("Down", 252)

glm.pred[glm.probs > .5] <- "Up"

table(glm.pred , Direction.2005)

##         Direction.2005
## glm.pred Down Up
##     Down   77 97
##     Up     34 44

# Mean equal to Direction.2005
mean(glm.pred == Direction.2005)

## [1] 0.4801587

# Mean not equal to Direction.2005
mean(glm.pred != Direction.2005)

## [1] 0.5198413
```

The results are rather disappointing : the test error rate is 52%, which is worse than random guessing

We recall that the logistic regression model had very underwhelming p-values associated with all of the predictors, and that the smallest p-value, though not very small, corresponded to *Lag1*. Perhaps by removing the variables that appear not to be helpful in predicting Direction, we can obtain a more effective model.

```
glm.fits <- glm (Direction ~ Lag1 + Lag2 , data = Smarket ,
family = binomial , subset = train)

glm.probs <- predict (glm.fits , Smarket.2005,
type = "response")

glm.pred <- rep("Down", 252)

glm.pred[glm.probs > 0.5] <- "Up"

table (glm.pred , Direction.2005)

##         Direction.2005
## glm.pred Down  Up
##     Down   35  35
##     Up     76 106
```

```r
mean (glm.pred == Direction.2005)

## [1] 0.5595238

106 / (106 + 76)

## [1] 0.5824176
```

Now the results appear to be a little better : $56% $of the daily movements have been correctly predicted. It is worth noting that in this case, a much simpler strategy of predicting that the market will increase every day will also be correct 56% of the time. Hence, in terms of overall error rate, the *logistic regression* method is *no better* than the *naive* approach. However, the *confusion matrix* shows that on days when *logistic regression predicts* an increase in the market, it has a 58% accuracy rate.

Suppose that we want to predict the returns associated with particular values of `Lag1` and `Lag2`. In particular, we want to predict Direction on a day when `Lag1` and `Lag2` equal *1.2* and *1.1*, respectively, and on a *day* when they equal *1.5* and *−0.8*. We do this using the `predict()` function .

```r
predict(glm.fits, newdata = data.frame(Lag1 = c(1.2, 1.5),
                                       Lag2 = c(1.1, -0.8)), type =
"response")

##         1         2
## 0.4791462 0.4960939
```

1.6.2 Linear Discriminant Analysis (LDA)

Now we will perform *LDA* on the *Smarket* data. In R, we fit an LDA model using the *lda*( ) function, which is part of the *MASS* library. Notice that the syntax for the `lda()` function is identical to that of `lm()`, and to that of `glm()` except for the absence of the family option. We fit the model using only the observations before 2005.

```r
library (MASS)

lda.fit <- lda(Direction ~ Lag1 + Lag2 , data = Smarket , subset = train)

lda.fit

## Call:
## lda(Direction ~ Lag1 + Lag2, data = Smarket, subset = train)
##
## Prior probabilities of groups:
##     Down       Up
## 0.491984 0.508016
##
## Group means:
##             Lag1        Lag2
## Down  0.04279022  0.03389409
## Up   -0.03954635 -0.03132544
```

```
##
## Coefficients of linear discriminants:
##             LD1
## Lag1 -0.6420190
## Lag2 -0.5135293
```

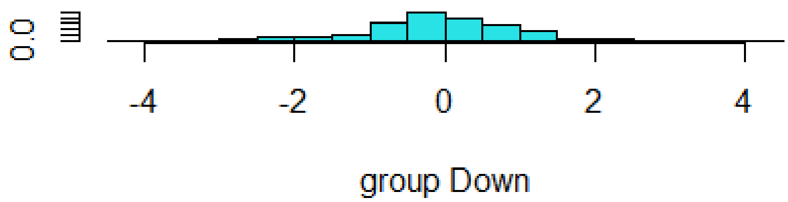
$\widehat{\pi_\iota}$ is *Prior probabilities of groups* The *LDA* output indicates that $\widehat{\pi_1} = 0.492$ and $\widehat{\pi_2} = 0.508$; in other words,49.2% of the training observations correspond to days during which the market went down.

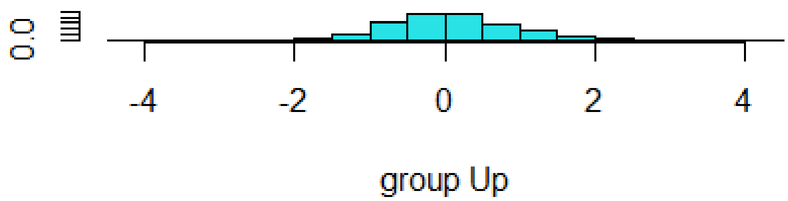
It also provides the group means ; these are the average of each predictor within each class, and are used by LDA as estimates of μ_k. These suggest that there is a tendency for the previous 2 days' returns to be negative on days when the market increases, and a tendency for the previous days' returns to be positive on days when the market declines.

The *coefficients of linear discriminants* output provides the linear combination of *Lag1* and *Lag2* that are used to form the LDA decision rule. In other words, these are the multipliers of the elements of $X = x$. If $-0.642 \times Lag1 - 0.514 \times Lag2$ is large, then the *LDA* classifier will predict a market increase and if it is small, then the *LDA* classifier will predict a market decline.

The $plot()$ function produces plots of the *linear discriminants*, obtained by computing $-0.642 \times Lag1 - 0.514 \times Lag2$ for each of the training observations. The Up and Down observations are displayed separately.

```
# Plot
plot(lda.fit)
```





The *predict*( ) function returns a list with three elements. The first element, *class*, contains `LDA's predictions` about the movement of the market. The second element, *posterior*, is a *matrix* whose k^{th} column contains the `posterior probability` that the corresponding observation belongs to the k^{th} class Finally, x contains the *linear discriminants*, described earlier.

```
lda.pred <- predict(lda.fit , Smarket.2005)

names (lda.pred)

## [1] "class"     "posterior" "x"
```

The *LDA* and *logistic regression* predictions are almost identical.

```
lda.class <- lda.pred$class

# Table
table(lda.class, Direction.2005)

##            Direction.2005
## lda.class Down  Up
##      Down   35  35
##      Up     76 106

# Mean
mean(lda.class == Direction.2005)
```

```
## [1] 0.5595238
```

Applying a 50% threshold to the posterior probabilities allows us to recreate the predictions contained in *lda.pred$class*.

```
sum(lda.pred$posterior[, 1] >= 0.5)

## [1] 70

sum (lda.pred$posterior[, 1] < 0.5)

## [1] 182
```

Notice that the posterior probability output by the model corresponds to the probability that the market will decrease :

```
lda.pred$posterior[1:20, 1]

##       999      1000      1001      1002      1003      1004 
## 0.4901792 0.4792185 0.4668185 0.4740011 0.4927877 0.4938562 
##      1005      1006      1007      1008      1009      1010 
## 0.4951016 0.4872861 0.4907013 0.4844026 0.4906963 0.5119988 
##      1011      1012      1013      1014      1015      1016 
## 0.4895152 0.4706761 0.4744593 0.4799583 0.4935775 0.5030894 
##      1017      1018 
## 0.4978806 0.4886331

lda.class[1:20]

##  [1] Up   Up   Up   Up   Up   Up   Up   Up   Up   Up   Up   Down
## [13] Up   Up   Up   Up   Up   Down Up   Up  
## Levels: Down Up
```

If we wanted to use a posterior probability threshold other than 50% in order to make predictions, then we could easily do so. For instance, suppose that we wish to predict a market decrease only if we are very certain that the market will indeed decrease on that day—say, if the posterior probability is at least 90% .

```
sum (lda.pred$posterior[, 1] > 0.9)

## [1] 0
```

No days in 2005 meet that threshold! In fact, the greatest posterior probability of decrease in all of 2005 was 52.02%

1.6.3 Quadratic Discriminant Analysis

We will now fit a *QDA* model to the *Smarket* data. QDA is implemented in R using the *qda*( ) function, which is also part of the *MASS* library. The `qda()` syntax is identical to that of `lda()` .

```
qda.fit <- qda (Direction ~ Lag1 + Lag2 , data = Smarket ,
subset = train)
```

```
qda.fit

## Call:
## qda(Direction ~ Lag1 + Lag2, data = Smarket, subset = train)
##
## Prior probabilities of groups:
##     Down       Up
## 0.491984 0.508016
##
## Group means:
##             Lag1        Lag2
## Down  0.04279022  0.03389409
## Up   -0.03954635 -0.03132544
```

The output contains the group means. But it does not contain the coefficients of the *linear discriminants*, because the *QDA* classifier involves a quadratic, rather than a linear, function of the predictors. The `predict()` function works in exactly the same fashion as for *LDA*.

```
qda.class <- predict (qda.fit , Smarket.2005)$class

table (qda.class , Direction.2005)

##          Direction.2005
## qda.class Down  Up
##      Down   30  20
##      Up     81 121

mean (qda.class == Direction.2005)

## [1] 0.5992063
```

Interestingly, the *QDA predictions* are accurate almost 60% of the time, even though the 2005 data was not used to fit the model. This level of accuracy is quite impressive for stock market data, which is known to be quite hard to model accurately. This suggests that the *quadratic form* assumed by QDA may capture the true relationship more accurately than the linear forms assumed by *LDA* and *logistic regression*. However, we recommend evaluating this method's performance on a larger test set before betting that this approach will consistently beat the market!

1.6.4 Naive Bayes

Next, we fit a *naive Bayes* model to the *Smarket* data. Naive Bayes is implemented in R using the $naiveBayes(\quad)$ function, which is part of the $e1071$ `naiveBayes()` library. The syntax is identical to that of `lda()` and `qda()`. By default, this implementation of the naive Bayes classifier models each quantitative feature using a *Gaussian distribution*. However, a kernel density method can also be used to estimate the distributions.

```
library(e1071)
```

```r
nb.fit <- naiveBayes (Direction ~ Lag1 + Lag2 , data = Smarket , subset = train)

nb.fit
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##     Down       Up
## 0.491984 0.508016
##
## Conditional probabilities:
##       Lag1
## Y             [,1]     [,2]
##   Down  0.04279022 1.227446
##   Up   -0.03954635 1.231668
##
##       Lag2
## Y             [,1]     [,2]
##   Down  0.03389409 1.239191
##   Up   -0.03132544 1.220765
```

The output contains the *estimated mean* and *standard deviation* for each variable in each class. For example, the *mean* for *Lag1* is 0.0428 for $Direction = Down$, and the *standard deviation* is 1.23 . We can easily verify this :

```r
mean(Lag1[train][Direction[train] == "Down"])

## [1] 0.04279022

sd(Lag1[train][Direction[train] == "Down"])

## [1] 1.227446
```

The `predict()` function is straight-forward.

```r
nb.class <- predict(nb.fit, Smarket.2005)

table(nb.class, Direction.2005)

##          Direction.2005
## nb.class Down  Up
##     Down   28  20
##     Up     83 121

mean(nb.class == Direction.2005)
```

```
## [1] 0.5912698
```

Naive Bayes performs very well on this data, with accurate predictions over 59% of the time. This is slightly worse than *QDA*, but much better than *LDA*.

The `predict()` function can also generate estimates of the probability that each observation belongs to a particular class.

```
nb.preds <- predict(nb.fit, Smarket.2005, type = "raw")

nb.preds[1:5 , ]

##           Down        Up
## [1,] 0.4873164 0.5126836
## [2,] 0.4762492 0.5237508
## [3,] 0.4653377 0.5346623
## [4,] 0.4748652 0.5251348
## [5,] 0.4901890 0.5098110
```

1.6.5 K-Nearest Neighbors

We will now perform **KNN** using the $knn(\quad)$ function, which is part of the *class* library. This function works rather differently from the other model fitting functions that we have encountered thus far. Rather than a two-step approach in which we first fit the model and then we use the model to make predictions, `knn()` forms predictions using a single command. The function requires four inputs.

1. A matrix containing the predictors associated with the training data, labeled $train.X$ below.

2. A matrix containing the predictors associated with the data for which we wish to make predictions, labeled $test.X$ below.

3. A vector containing the class labels for the training observations, labeled $train.Direction$ below.

4. A value for K, the number of nearest neighbors to be used by the classifier.

We use the `cbind()` function, short for column bind, to bind the *Lag1* and *cbind() Lag2* variables together into two matrices, one for the training set and the other for the test set .

```
library (class)
train.X <- cbind(Lag1 , Lag2)[train , ]

test.X <- cbind(Lag1 , Lag2)[!train , ]

train.Direction <- Direction[train]
```

Now the `knn()` function can be used to predict the market's movement for the dates in 2005. We set a random seed before we apply *knn()* because if several observations are tied as nearest neighbors, then R will randomly break the tie. Therefore, a seed must be set in order to ensure reproducibility of results.

```
set.seed(1)

knn.pred <- knn(train.X, test.X, train.Direction, k = 1)

table(knn.pred, Direction.2005)

##          Direction.2005
## knn.pred Down Up
##     Down   43 58
##     Up     68 83

(83 + 43) / 252

## [1] 0.5
```

The results using $K = 1$ are not very good, since only 50% of the observations are correctly predicted. Of course, it may be that $K = 1$ results in an overly flexible fit to the data. Below, we repeat the analysis using $K = 3$.

```
knn.pred <- knn(train.X, test.X, train.Direction, k = 3)

table(knn.pred, Direction.2005)

##          Direction.2005
## knn.pred Down Up
##     Down   48 54
##     Up     63 87

mean(knn.pred == Direction.2005)

## [1] 0.5357143
```

The results have improved slightly. But increasing K further turns out to provide no further improvements. It appears that for this data, **QDA** provides the best results of the methods that we have examined so far.

KNN does not perform well on the S*market* data but it does often provide impressive results. As an example we will apply the KNN approach to the **Caravan** data set, which is part of the **ISLR2** library. This data set includes 85 predictors that measure demographic characteristics for 5,822 individuals. The response variable is Purchase, which indicates whether or not a given individual p*urchases* a *caravan insurance policy*. In this data set, only **6%** of people purchased caravan insurance.

```
data("Caravan")

dim(Caravan)

## [1] 5822   86
```

```
attach (Caravan)

summary (Purchase)

##   No  Yes
## 5474  348

348/5822

## [1] 0.05977327
```

Because the *KNN* classifier predicts the class of a given test observation by identifying the observations that are nearest to it, the scale of the variables matters. Variables that are on a large scale will have a much larger effect on the *distance* between the observations, and hence on the KNN classifier, than variables that are on a small scale. For instance, imagine a data set that contains two variables, *salary* and *age* (measured in dollars and years, respectively).

A good way to handle this problem is to *standardize* the data so that all standardize variables are given a *mean of zero* and a *standard deviation of one*. Then all variables will be on a comparable scale. The *scale*(   ) function does just this. In standardizing the data, we *exclude column 86*, because that is the qualitative *Purchase* variable.

```
standardized.X <- scale(Caravan[, -86])

var(Caravan[, 1])

## [1] 165.0378

var(Caravan[, 2])

## [1] 0.1647078

var(standardized.X[, 1])

## [1] 1

var(standardized.X[, 2])

## [1] 1
```

Now every column of standardized.X has a standard deviation of one and a mean of zero.

We now split the observations into a test set, containing the first 1,000 observations, and a training set, containing the remaining observations. We fit a KNN model on the training data using *K = 1*, and evaluate its performance on the test data.

```
test <- 1:1000

train.X <- standardized.X[-test , ]

test.X <- standardized.X[test , ]
```

```r
train.Y <-Caravan$Purchase[-test]

test.Y <-  Caravan$Purchase[test]

set.seed(1)

knn.pred <- knn(train.X, test.X, train.Y, k = 1)

mean(test.Y != knn.pred)

## [1] 0.118

mean(test.Y != "No")

## [1] 0.059
```

The vector *test* is numeric, with values from 1 through 1, 000. Typing *standardized.X[test,]* yields the submatrix of the data containing the observations whose indices range from 1 to 1,000, whereas typing *standardized.X[-test,]* yields the submatrix containing the observations whose indices do not range from 1 to 1,000. The *KNN* error rate on the 1,000 test observations is just under *12%*. At first glance, this may appear to be fairly good. However, since only *6%* of customers purchased insurance, we could get the error rate down to *6%* by always predicting No regardless of the values of the predictors

It turns out that *KNN* with *K=1* does far better than random guessing among the customers that are predicted to buy insurance.

```r
table(knn.pred, test.Y)

##         test.Y
## knn.pred  No Yes
##      No  873  50
##      Yes  68   9

9/(68+9)

## [1] 0.1168831
```

Among *77* such customers, *9, or 11.7 %*, actually do purchase insurance. This is double the rate that one would obtain from random guessing.

Using *K=3*, the success rate increases to *19%*, and with *K=5* the rate is *26.7%*. This is over four times the rate that results from random guessing. It appears that *KNN* is finding some real patterns in a difficult data set!

```r
knn.pred <- knn(train.X, test.X, train.Y, k = 3)

table(knn.pred , test.Y)
```

```
##         test.Y
## knn.pred  No Yes
##      No  920  54
##      Yes  21   5

5/26

## [1] 0.1923077

knn.pred <- knn(train.X, test.X, train.Y, k = 5)

table(knn.pred , test.Y)

##         test.Y
## knn.pred  No Yes
##      No  930  55
##      Yes  11   4

4/15

## [1] 0.2666667
```

However, while this strategy is cost-effective, it is worth noting that only *15* customers are predicted to purchase insurance using *KNN* with *K=5.* In practice, the insurance company may wish to expend resources on convincing more than just *15* potential customers to buy insurance.

As a comparison, we can also fit a *logistic regression model* to the data. If we use *0.5* as the predicted probability cut-off for the classifier, then we have a problem: only seven of the test observations are predicted to purchase insurance. However, we are not required to use a cut-off of *0.5*. If we instead predict a purchase any time the predicted probability of purchase exceeds *0.25*, we get much better results: we predict that *33* people will purchase insurance, and we are correct for about *33%* of these people. This is over five times better than random guessing

```
# glm.fits <- glm (Purchase ~ ., data = Caravan , family = binomial , subset
= -test)
# Warning message: glm.fits: fitted probabilities numerically 0 or 1 occurred

glm.probs <- predict(glm.fits , Caravan[test , ], type = "response")

## Warning: 'newdata' had 1000 rows but variables found have 1250
## rows

glm.pred <- rep("No", 1000)

glm.pred[glm.probs > 0.5] <- " Yes "

#table(glm.pred , test.Y)

glm.pred <- rep("No", 1000)
```

```
glm.pred[glm.probs > 0.25] <- " Yes "

# table(glm.pred , test.Y)

11 / (22 + 11)

## [1] 0.3333333
```

1.6.6 Poisson Regression

Finally, we fit a **Poisson Regression Model** to the *Bikeshare* data set, which measures the number of bike rentals (*bikers*) per hour in Washington, DC. The data can be found in the *ISLR2* library.

```
data("Bikeshare")

dim(Bikeshare)

## [1] 8645   15

names(Bikeshare)

##  [1] "season"     "mnth"       "day"        "hr"
##  [5] "holiday"    "weekday"    "workingday" "weathersit"
##  [9] "temp"       "atemp"      "hum"        "windspeed"
## [13] "casual"     "registered" "bikers"
```

We begin by fitting a least squares *linear regression model* to the data.

```
mod.lm <- lm(bikers ~ mnth + hr + workingday + temp + weathersit , data =
Bikeshare)

summary(mod.lm)

##
## Call:
## lm(formula = bikers ~ mnth + hr + workingday + temp + weathersit,
##     data = Bikeshare)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -299.00  -45.70   -6.23   41.08  425.29
##
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)                -68.632      5.307 -12.932  < 2e-16
## mnthFeb                      6.845      4.287   1.597 0.110398
## mnthMarch                   16.551      4.301   3.848 0.000120
## mnthApril                   41.425      4.972   8.331  < 2e-16
## mnthMay                     72.557      5.641  12.862  < 2e-16
```

```
## mnthJune                   67.819      6.544  10.364  < 2e-16
## mnthJuly                   45.324      7.081   6.401 1.63e-10
## mnthAug                    53.243      6.640   8.019 1.21e-15
## mnthSept                   66.678      5.925  11.254  < 2e-16
## mnthOct                    75.834      4.950  15.319  < 2e-16
## mnthNov                    60.310      4.610  13.083  < 2e-16
## mnthDec                    46.458      4.271  10.878  < 2e-16
## hr1                       -14.579      5.699  -2.558 0.010536
## hr2                       -21.579      5.733  -3.764 0.000168
## hr3                       -31.141      5.778  -5.389 7.26e-08
## hr4                       -36.908      5.802  -6.361 2.11e-10
## hr5                       -24.135      5.737  -4.207 2.61e-05
## hr6                        20.600      5.704   3.612 0.000306
## hr7                       120.093      5.693  21.095  < 2e-16
## hr8                       223.662      5.690  39.310  < 2e-16
## hr9                       120.582      5.693  21.182  < 2e-16
## hr10                       83.801      5.705  14.689  < 2e-16
## hr11                      105.423      5.722  18.424  < 2e-16
## hr12                      137.284      5.740  23.916  < 2e-16
## hr13                      136.036      5.760  23.617  < 2e-16
## hr14                      126.636      5.776  21.923  < 2e-16
## hr15                      132.087      5.780  22.852  < 2e-16
## hr16                      178.521      5.772  30.927  < 2e-16
## hr17                      296.267      5.749  51.537  < 2e-16
## hr18                      269.441      5.736  46.976  < 2e-16
## hr19                      186.256      5.714  32.596  < 2e-16
## hr20                      125.549      5.704  22.012  < 2e-16
## hr21                       87.554      5.693  15.378  < 2e-16
## hr22                       59.123      5.689  10.392  < 2e-16
## hr23                       26.838      5.688   4.719 2.41e-06
## workingday                  1.270      1.784   0.711 0.476810
## temp                      157.209     10.261  15.321  < 2e-16
## weathersitcloudy/misty    -12.890      1.964  -6.562 5.60e-11
## weathersitlight rain/snow  -66.494      2.965 -22.425  < 2e-16
## weathersitheavy rain/snow -109.745     76.667  -1.431 0.152341
##
## (Intercept)               ***
## mnthFeb
## mnthMarch                 ***
## mnthApril                 ***
## mnthMay                   ***
## mnthJune                  ***
## mnthJuly                  ***
## mnthAug                   ***
## mnthSept                  ***
## mnthOct                   ***
## mnthNov                   ***
## mnthDec                   ***
## hr1                       *
## hr2                       ***
```

```
## hr3                        ***
## hr4                        ***
## hr5                        ***
## hr6                        ***
## hr7                        ***
## hr8                        ***
## hr9                        ***
## hr10                       ***
## hr11                       ***
## hr12                       ***
## hr13                       ***
## hr14                       ***
## hr15                       ***
## hr16                       ***
## hr17                       ***
## hr18                       ***
## hr19                       ***
## hr20                       ***
## hr21                       ***
## hr22                       ***
## hr23                       ***
## workingday
## temp                       ***
## weathersitcloudy/misty     ***
## weathersitlight rain/snow ***
## weathersitheavy rain/snow
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 76.5 on 8605 degrees of freedom
## Multiple R-squared:  0.6745, Adjusted R-squared:  0.6731
## F-statistic: 457.3 on 39 and 8605 DF,  p-value: < 2.2e-16
```

Due to space constraints, we truncate the output of `summary(mod.lm)`. In mod.lm, the first level of *hr(0)* and *mnth(Jan)* are treated as the baseline values, and so no coefficient estimates are provided for them: implicitly, their coefficient estimates are zero, and all other levels are measured relative to these baselines. For example, the *Feb* coefficient of *6.845* signifies that, holding all other variables constant, there are on average about *7* more riders in February than in January. Similarly there are about *16.5* more riders in March than in January.

Different coding of the variables hr and mnth, as follows :

```
contrasts(Bikeshare$hr) = contr.sum(24)

contrasts(Bikeshare$mnth) = contr.sum (12)

mod.lm2 <- lm(bikers ~ mnth + hr + workingday + temp + weathersit , data =
Bikeshare)
```

```
summary (mod.lm2)

##
## Call:
## lm(formula = bikers ~ mnth + hr + workingday + temp + weathersit,
##     data = Bikeshare)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -299.00  -45.70   -6.23   41.08  425.29
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 73.5974     5.1322  14.340  < 2e-16
## mnth1                      -46.0871     4.0855 -11.281  < 2e-16
## mnth2                      -39.2419     3.5391 -11.088  < 2e-16
## mnth3                      -29.5357     3.1552  -9.361  < 2e-16
## mnth4                       -4.6622     2.7406  -1.701  0.08895
## mnth5                       26.4700     2.8508   9.285  < 2e-16
## mnth6                       21.7317     3.4651   6.272 3.75e-10
## mnth7                       -0.7626     3.9084  -0.195  0.84530
## mnth8                        7.1560     3.5347   2.024  0.04295
## mnth9                       20.5912     3.0456   6.761 1.46e-11
## mnth10                      29.7472     2.6995  11.019  < 2e-16
## mnth11                      14.2229     2.8604   4.972 6.74e-07
## hr1                        -96.1420     3.9554 -24.307  < 2e-16
## hr2                       -110.7213     3.9662 -27.916  < 2e-16
## hr3                       -117.7212     4.0165 -29.310  < 2e-16
## hr4                       -127.2828     4.0808 -31.191  < 2e-16
## hr5                       -133.0495     4.1168 -32.319  < 2e-16
## hr6                       -120.2775     4.0370 -29.794  < 2e-16
## hr7                        -75.5424     3.9916 -18.925  < 2e-16
## hr8                         23.9511     3.9686   6.035 1.65e-09
## hr9                        127.5199     3.9500  32.284  < 2e-16
## hr10                        24.4399     3.9360   6.209 5.57e-10
## hr11                       -12.3407     3.9361  -3.135  0.00172
## hr12                         9.2814     3.9447   2.353  0.01865
## hr13                        41.1417     3.9571  10.397  < 2e-16
## hr14                        39.8939     3.9750  10.036  < 2e-16
## hr15                        30.4940     3.9910   7.641 2.39e-14
## hr16                        35.9445     3.9949   8.998  < 2e-16
## hr17                        82.3786     3.9883  20.655  < 2e-16
## hr18                       200.1249     3.9638  50.488  < 2e-16
## hr19                       173.2989     3.9561  43.806  < 2e-16
## hr20                        90.1138     3.9400  22.872  < 2e-16
## hr21                        29.4071     3.9362   7.471 8.74e-14
## hr22                        -8.5883     3.9332  -2.184  0.02902
## hr23                       -37.0194     3.9344  -9.409  < 2e-16
## workingday                   1.2696     1.7845   0.711  0.47681
```

```
## temp                         157.2094    10.2612  15.321  < 2e-16
## weathersitcloudy/misty       -12.8903     1.9643  -6.562 5.60e-11
## weathersitlight rain/snow    -66.4944     2.9652 -22.425  < 2e-16
## weathersitheavy rain/snow   -109.7446    76.6674  -1.431  0.15234
##
## (Intercept)                 ***
## mnth1                       ***
## mnth2                       ***
## mnth3                       ***
## mnth4                       .
## mnth5                       ***
## mnth6                       ***
## mnth7
## mnth8                       *
## mnth9                       ***
## mnth10                      ***
## mnth11                      ***
## hr1                         ***
## hr2                         ***
## hr3                         ***
## hr4                         ***
## hr5                         ***
## hr6                         ***
## hr7                         ***
## hr8                         ***
## hr9                         ***
## hr10                        ***
## hr11                        **
## hr12                        *
## hr13                        ***
## hr14                        ***
## hr15                        ***
## hr16                        ***
## hr17                        ***
## hr18                        ***
## hr19                        ***
## hr20                        ***
## hr21                        ***
## hr22                        *
## hr23                        ***
## workingday
## temp                        ***
## weathersitcloudy/misty      ***
## weathersitlight rain/snow   ***
## weathersitheavy rain/snow
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 76.5 on 8605 degrees of freedom
```

```
## Multiple R-squared:  0.6745, Adjusted R-squared:  0.6731
## F-statistic: 457.3 on 39 and 8605 DF,  p-value: < 2.2e-16
```

What is the difference between the two codings ? In *mod.lm2*, a coefficient estimate is reported for all but the last level of *hr* and *mnth*. Importantly, in *mod.lm2*, the coefficient estimate for the last level of mnth is not zero: instead, it equals the *negative of the sum of the coefficient estimates for all of the other levels*. Similarly, in *mod.lm2*, the coefficient estimate for the last level of hr is the negative of the sum of the coefficient estimates for all of the other levels. This means that the coefficients of hr and mnth in *mod.lm2* will always sum to zero, and can be interpreted as the difference from the mean level. For example, the coefficient for *January of −46.087* indicates that, holding all other variables constant, there are typically 46 fewer riders in January relative to the yearly average.

It is important to realize that the choice of coding really does not matter, provided that we interpret the model output correctly in light of the coding used. For example, we see that the predictions from the linear model are the same regardless of coding :

```
sum((predict(mod.lm) - predict(mod.lm2))^2)

## [1] 1.426274e-18
```

The sum of squared differences is zero. We can also see this using the $all.equal(\quad)$ function :

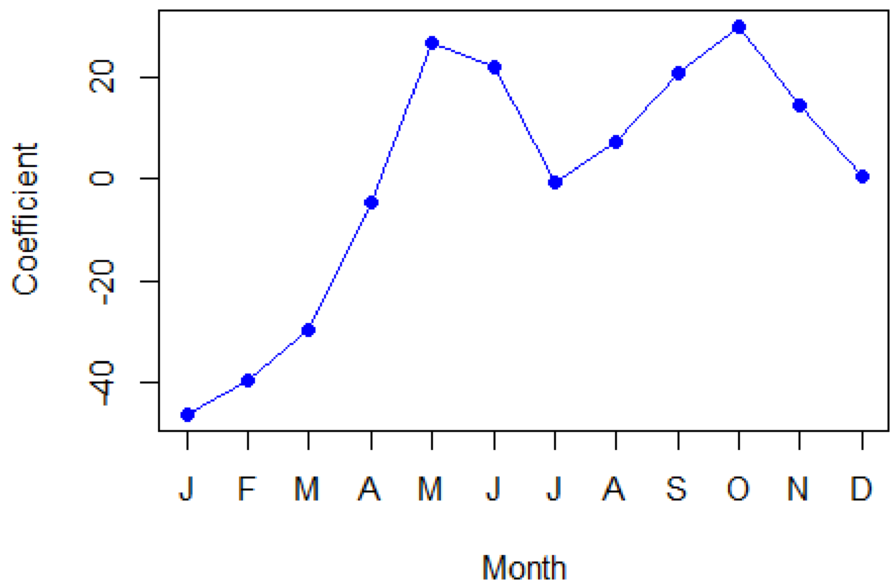
```
all.equal(predict (mod.lm), predict (mod.lm2))

## [1] TRUE
```

The coefficients for January through November can be obtained directly from the mod.lm2 object. The coefficient for December must be explicitly computed as the negative sum of all the other months

```
coef.months <- c(coef (mod.lm2)[2:12],
                  -sum ( coef (mod.lm2)[2:12]))
```

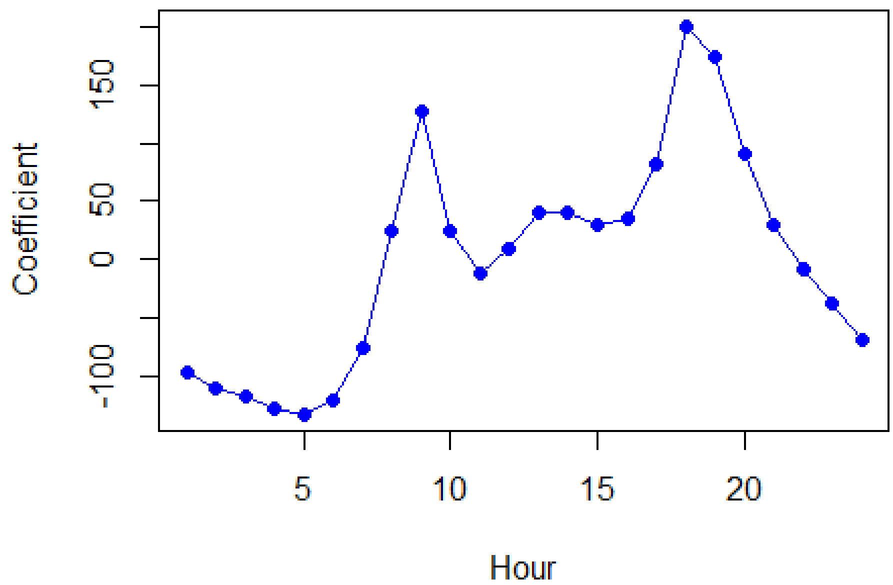
To make the plot, we manually label the x-axis with the names of the months.

```
plot (coef.months , xlab = " Month ", ylab = " Coefficient ",
xaxt = "n", col = " blue ", pch = 19, type = "o")

axis(side = 1, at = 1:12, labels = c("J", "F", "M", "A",
                                     "M", "J", "J", "A",
                                     "S","O", "N", "D"))
```



Reproducing the right-hand side of Figure 4.13 follows a similar process.

```
coef.hours <- c(coef (mod.lm2)[13:35],
                 -sum ( coef (mod.lm2)[13:35]))

plot(coef.hours , xlab = " Hour ", ylab = " Coefficient ",
     col = " blue ", pch = 19, type = "o")
```



Now, we consider instead fitting a *Poisson regression* model to the *Bikeshare* data. Very little changes, except that we now use the function $glm(\quad)$ with the argument $family = poisson$ to specify that we wish to fit a *Poisson regression model* :

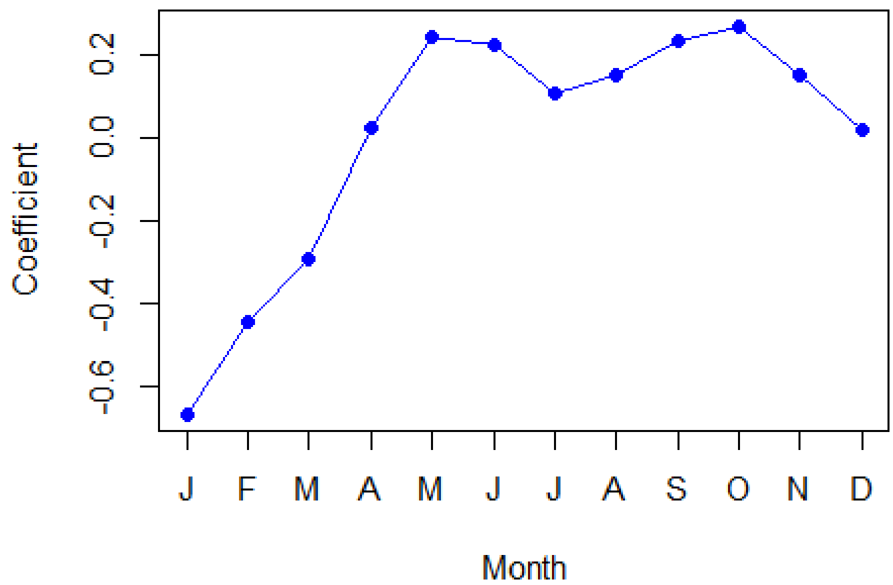
```
mod.pois <- glm(bikers ~ mnth + hr + workingday + temp + weathersit ,
                data = Bikeshare , family = poisson)

summary(mod.pois)

##
## Call:
## glm(formula = bikers ~ mnth + hr + workingday + temp + weathersit,
##     family = poisson, data = Bikeshare)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -20.7574   -3.3441   -0.6549    2.6999   21.9628
##
## Coefficients:
##                            Estimate Std. Error  z value Pr(>|z|)
## (Intercept)                4.118245   0.006021  683.964  < 2e-16
## mnth1                     -0.670170   0.005907 -113.445  < 2e-16
## mnth2                     -0.444124   0.004860  -91.379  < 2e-16
## mnth3                     -0.293733   0.004144  -70.886  < 2e-16
## mnth4                      0.021523   0.003125    6.888 5.66e-12
## mnth5                      0.240471   0.002916   82.462  < 2e-16
## mnth6                      0.223235   0.003554   62.818  < 2e-16
```

```
## mnth7                       0.103617   0.004125   25.121  < 2e-16
## mnth8                       0.151171   0.003662   41.281  < 2e-16
## mnth9                       0.233493   0.003102   75.281  < 2e-16
## mnth10                      0.267573   0.002785   96.091  < 2e-16
## mnth11                      0.150264   0.003180   47.248  < 2e-16
## hr1                        -0.754386   0.007879  -95.744  < 2e-16
## hr2                        -1.225979   0.009953 -123.173  < 2e-16
## hr3                        -1.563147   0.011869 -131.702  < 2e-16
## hr4                        -2.198304   0.016424 -133.846  < 2e-16
## hr5                        -2.830484   0.022538 -125.586  < 2e-16
## hr6                        -1.814657   0.013464 -134.775  < 2e-16
## hr7                        -0.429888   0.006896  -62.341  < 2e-16
## hr8                         0.575181   0.004406  130.544  < 2e-16
## hr9                         1.076927   0.003563  302.220  < 2e-16
## hr10                        0.581769   0.004286  135.727  < 2e-16
## hr11                        0.336852   0.004720   71.372  < 2e-16
## hr12                        0.494121   0.004392  112.494  < 2e-16
## hr13                        0.679642   0.004069  167.040  < 2e-16
## hr14                        0.673565   0.004089  164.722  < 2e-16
## hr15                        0.624910   0.004178  149.570  < 2e-16
## hr16                        0.653763   0.004132  158.205  < 2e-16
## hr17                        0.874301   0.003784  231.040  < 2e-16
## hr18                        1.294635   0.003254  397.848  < 2e-16
## hr19                        1.212281   0.003321  365.084  < 2e-16
## hr20                        0.914022   0.003700  247.065  < 2e-16
## hr21                        0.616201   0.004191  147.045  < 2e-16
## hr22                        0.364181   0.004659   78.173  < 2e-16
## hr23                        0.117493   0.005225   22.488  < 2e-16
## workingday                  0.014665   0.001955    7.502 6.27e-14
## temp                        0.785292   0.011475   68.434  < 2e-16
## weathersitcloudy/misty     -0.075231   0.002179  -34.528  < 2e-16
## weathersitlight rain/snow -0.575800   0.004058 -141.905  < 2e-16
## weathersitheavy rain/snow -0.926287   0.166782   -5.554 2.79e-08
##
## (Intercept)                ***
## mnth1                      ***
## mnth2                      ***
## mnth3                      ***
## mnth4                      ***
## mnth5                      ***
## mnth6                      ***
## mnth7                      ***
## mnth8                      ***
## mnth9                      ***
## mnth10                     ***
## mnth11                     ***
## hr1                        ***
## hr2                        ***
## hr3                        ***
## hr4                        ***
```

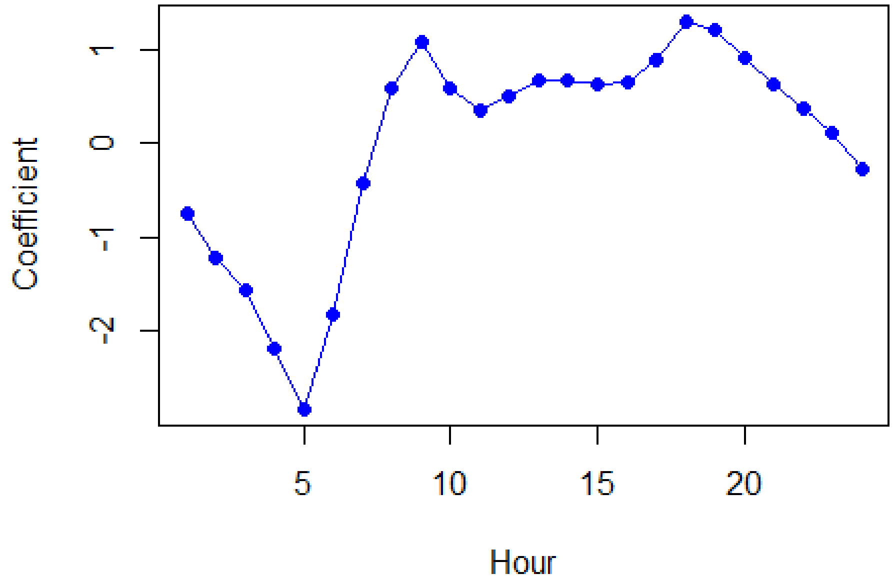
```
## hr5                       ***
## hr6                       ***
## hr7                       ***
## hr8                       ***
## hr9                       ***
## hr10                      ***
## hr11                      ***
## hr12                      ***
## hr13                      ***
## hr14                      ***
## hr15                      ***
## hr16                      ***
## hr17                      ***
## hr18                      ***
## hr19                      ***
## hr20                      ***
## hr21                      ***
## hr22                      ***
## hr23                      ***
## workingday                ***
## temp                      ***
## weathersitcloudy/misty    ***
## weathersitlight rain/snow ***
## weathersitheavy rain/snow ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 1052921  on 8644  degrees of freedom
## Residual deviance:  228041  on 8605  degrees of freedom
## AIC: 281159
##
## Number of Fisher Scoring iterations: 5
```

We can plot the coefficients associated with mnth and hr, in order to reproduce

```
coef.mnth <- c( coef (mod.pois)[2:12],
                -sum ( coef (mod.pois)[2:12]))

plot(coef.mnth , xlab = " Month ", ylab = " Coefficient ",
     xaxt = "n", col = " blue ", pch = 19, type = "o")

axis(side = 1, at = 1:12, labels = c("J", "F", "M", "A",
                                     "M", "J", "J", "A",
                                     "S", "O", "N", "D"))
```

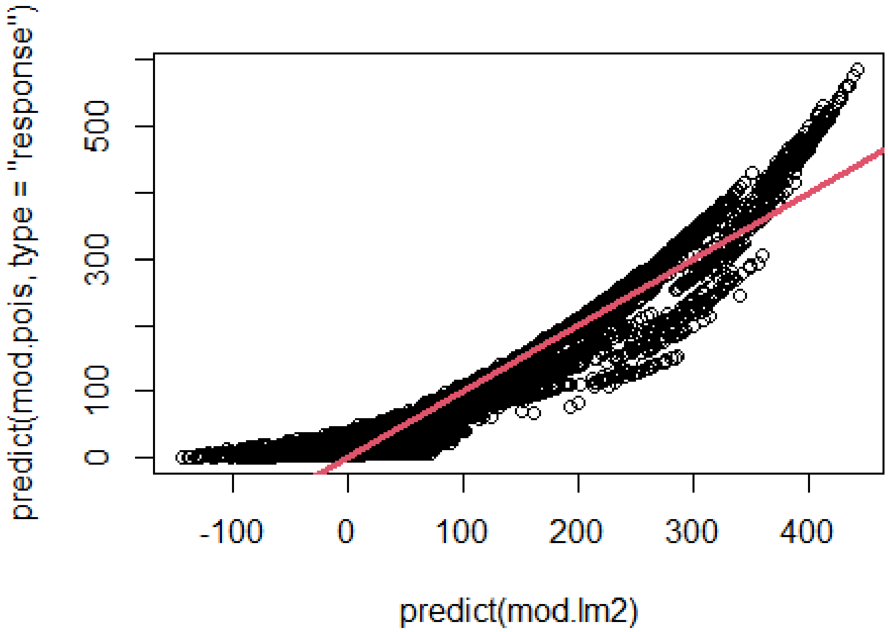



```
coef.hours <- c(coef(mod.pois)[13:35],
                -sum ( coef (mod.pois)[13:35]))

plot (coef.hours , xlab = " Hour ", ylab = " Coefficient ",
    col = " blue ", pch = 19, type = "o")
```



We can once again use the `predict()` function to obtain the fitted values (predictions) from this *Poisson regression model*. However, we must use the argument `type = "response"` to specify that we want R to output $e^{\hat{\beta_0} + \hat{\beta_1}X_1 + \hat{\beta_12X_2 + \cdots + \hat{\beta_p}X_p}$ rather than $\hat{\beta_0} + \hat{\beta_1}X_1 + \hat{\beta_12X_2 + \cdots + \hat{\beta_p}X_p$, which it will output by default.

```
plot(predict(mod.lm2),
     predict(mod.pois , type = "response"))

abline(0, 1, col = 2, lwd = 3)
```



The predictions from the Poisson regression model are correlated with those from the linear model; however, the former are non-negative. As a result the Poisson regression predictions tend to be larger than those from the linear model for either very low or very high levels of ridership.

In this section, we used the $glm(\quad)$ function with the argument $family = poisson$ in order to perform Poisson regression. Earlier in this lab we used the `glm()` function with $family = binomial$ to perform *logistic regression*. Other choices for the family argument can be used to fit other types of $GLMs$. For instance, $family = Gamma$ fits a *gamma regression model* .