Classification

YOUR NAME

This was adapted from James, Witten, Hastie, and Ibshirani (2021) An Introduction to Statistical Learning.

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, `lagone` through `lagfive`. 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.

```r
library(ISLR2)
names(Smarket)
```

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

```r
dim(Smarket)
```

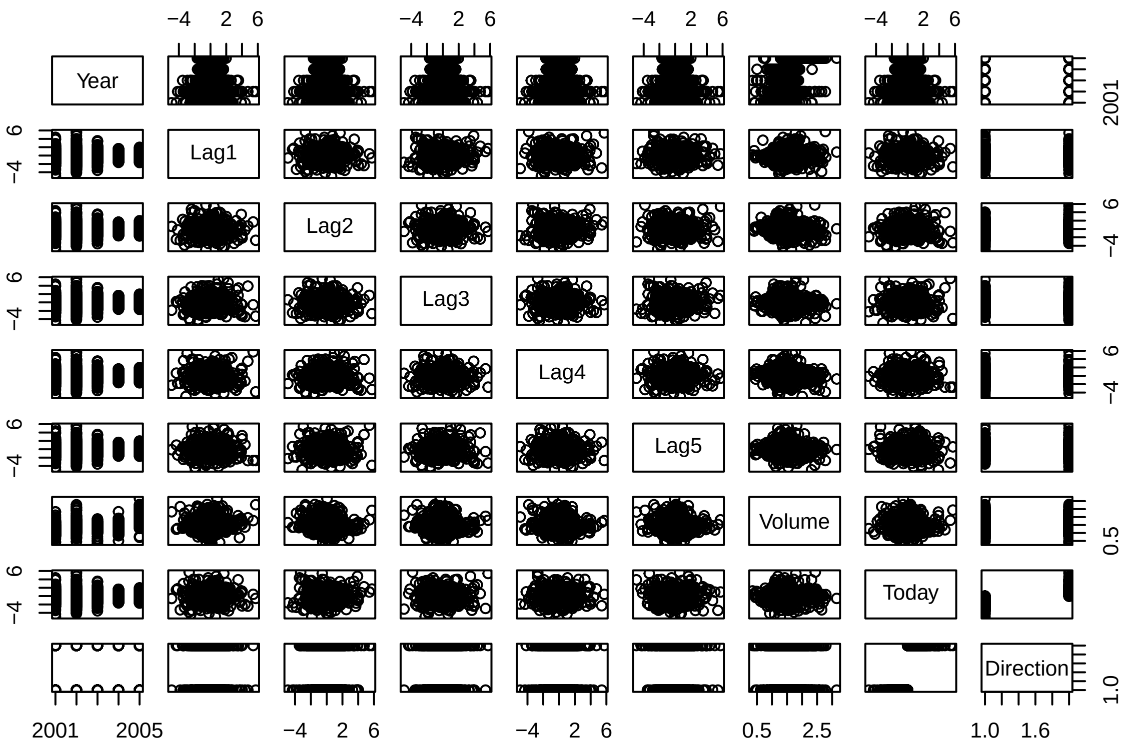
```
## [1] 1250    9
```

```r
summary(Smarket)
```

```
##       Year           Lag1                Lag2                Lag3
##  Min.   :2001   Min.   :-4.922000   Min.   :-4.922000   Min.   :-4.922000
##  1st Qu.:2002   1st Qu.:-0.639500   1st Qu.:-0.639500   1st Qu.:-0.640000
##  Median :2003   Median : 0.039000   Median : 0.039000   Median : 0.038500
##  Mean   :2003   Mean   : 0.003834   Mean   : 0.003919   Mean   : 0.001716
##  3rd Qu.:2004   3rd Qu.: 0.596750   3rd Qu.: 0.596750   3rd Qu.: 0.596750
##  Max.   :2005   Max.   : 5.733000   Max.   : 5.733000   Max.   : 5.733000
##       Lag4                Lag5              Volume           Today
##  Min.   :-4.922000   Min.   :-4.92200   Min.   :0.3561   Min.   :-4.922000
##  1st Qu.:-0.640000   1st Qu.:-0.64000   1st Qu.:1.2574   1st Qu.:-0.639500
##  Median : 0.038500   Median : 0.03850   Median :1.4229   Median : 0.038500
##  Mean   : 0.001636   Mean   : 0.00561   Mean   :1.4783   Mean   : 0.003138
##  3rd Qu.: 0.596750   3rd Qu.: 0.59700   3rd Qu.:1.6417   3rd Qu.: 0.596750
##  Max.   : 5.733000   Max.   : 5.73300   Max.   :3.1525   Max.   : 5.733000
##  Direction
##  Down:602
##  Up  :648
```

```
##
##
##
##
```

```r
pairs(Smarket)
```



The `cor()` function produces a matrix that contains all of the pairwise correlations among the predictors in a data set. The first command below gives an error message because the `direction` variable is qualitative.

```r
cor(Smarket)
```

```
## Error in cor(Smarket): 'x' must be numeric
```

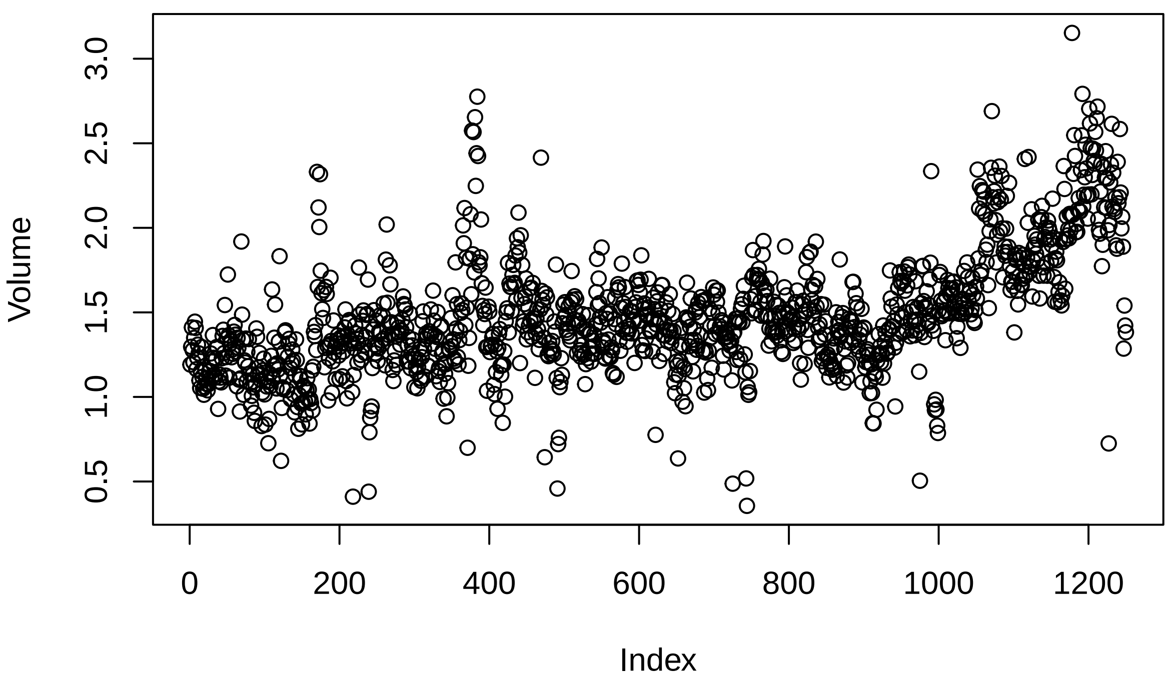
```r
cor(Smarket[, -9])
```

```
##              Year         Lag1         Lag2         Lag3         Lag4
## Year   1.00000000  0.029699649  0.030596422  0.033194581  0.035688718
## Lag1   0.02969965  1.000000000 -0.026294328 -0.010803402 -0.002985911
## Lag2   0.03059642 -0.026294328  1.000000000 -0.025896670 -0.010853533
## Lag3   0.03319458 -0.010803402 -0.025896670  1.000000000 -0.024051036
## Lag4   0.03568872 -0.002985911 -0.010853533 -0.024051036  1.000000000
## Lag5   0.02978799 -0.005674606 -0.003557949 -0.018808338 -0.027083641
## Volume 0.53900647  0.040909908 -0.043383215 -0.041823686 -0.048414246
## Today  0.03009523 -0.026155045 -0.010250033 -0.002447647 -0.006899527
```

```
##                Lag5      Volume        Today
## Year    0.029787995  0.53900647  0.030095229
## Lag1   -0.005674606  0.04090991 -0.026155045
## Lag2   -0.003557949 -0.04338321 -0.010250033
## Lag3   -0.018808338 -0.04182369 -0.002447647
## Lag4   -0.027083641 -0.04841425 -0.006899527
## Lag5    1.000000000 -0.02200231 -0.034860083
## Volume -0.022002315  1.00000000  0.014591823
## Today  -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.

```r
attach(Smarket)
plot(Volume)
```



Logistic Regression

Next, we will fit a logistic regression model in order to predict `direction` using `lagone` through `lagfive` 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.

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

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
##     Volume, family = binomial, data = Smarket)
##
## 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
```

The smallest p-value here is associated with `lagone`. 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 `lagone` and `direction`.

We use the `coef()` function in order to access just the coefficients for this fitted model. We can also use the `summary()` function to access particular aspects of the fitted model, such as the p-values for the coefficients.

```
coef(glm.fits)
```

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

```
summary(glm.fits)$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
```

```r
summary(glm.fits)$coef[, 4]
```

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

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. Here we have printed only the first ten probabilities. 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`.

```r
glm.probs <- predict(glm.fits, type = "response")
glm.probs[1:10]
```

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

```r
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 > .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
```

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

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

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

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

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. In this case, logistic regression correctly predicted the movement of the market 52.2 % 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, $100\% - 52.2\% = 47.8\%$, is the *training* error rate. As we have seen previously, the training error rate is often overly optimistic—it tends to underestimate the test 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.

```r
train <- (Year < 2005)
Smarket.2005 <- Smarket[!train, ]
dim(Smarket.2005)
```

```
## [1] 252   9
```

```r
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`. Boolean vectors can be used to obtain a subset of the rows or columns of a matrix. For instance, the command `Smarket[train, ]` would pick out a submatrix 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. That is, `!train` is a vector similar to `train`, except that the elements that are `TRUE` in `train` get swapped to `FALSE` in `!train`, and the elements that are `FALSE` in `train` get swapped to `TRUE` in `!train`. Therefore, `Smarket[!train, ]` yields a submatrix of the stock market data containing only the observations for which `train` is `FALSE`—that is, the observations with dates in 2005. The output above indicates that there are 252 such observations.

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.

```r
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.

```r
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
```

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

```
## [1] 0.4801587
```

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

```
## [1] 0.5198413
```

The `!=` notation means *not equal to*, and so the last command computes the test set error rate. The results are rather disappointing: the test error rate is 52 %, which is worse than random guessing! Of course this result is not all that surprising, given that one would not generally expect to be able to use previous days' returns to predict future market performance. (After all, if it were possible to do so, then the authors of this book would be out striking it rich rather than writing a statistics textbook.)

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 `lagone`. Perhaps by removing the variables that appear not to be helpful in predicting `direction`, we can obtain a more effective model. After all, using predictors that have no relationship with the response tends to cause a deterioration in the test error rate (since such predictors cause an increase in variance without a corresponding decrease in bias), and so removing such predictors may in turn yield an improvement. Below we have refit the logistic regression using just `lagone` and `lagtwo`, which seemed to have the highest predictive power in the original logistic regression model.

```r
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 > .5] <- "Up"
table(glm.pred, Direction.2005)
```

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

```
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. This suggests a possible trading strategy of buying on days when the model predicts an increasing market, and avoiding trades on days when a decrease is predicted. Of course one would need to investigate more carefully whether this small improvement was real or just due to random chance.

Suppose that we want to predict the returns associated with particular values of `lagone` and `lagtwo`. In particular, we want to predict `direction` on a day when `lagone` and `lagtwo` 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.

```
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
```

Linear Discriminant Analysis

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.

```
library(MASS)
```

```
## Warning: package 'MASS' was built under R version 4.4.1
```

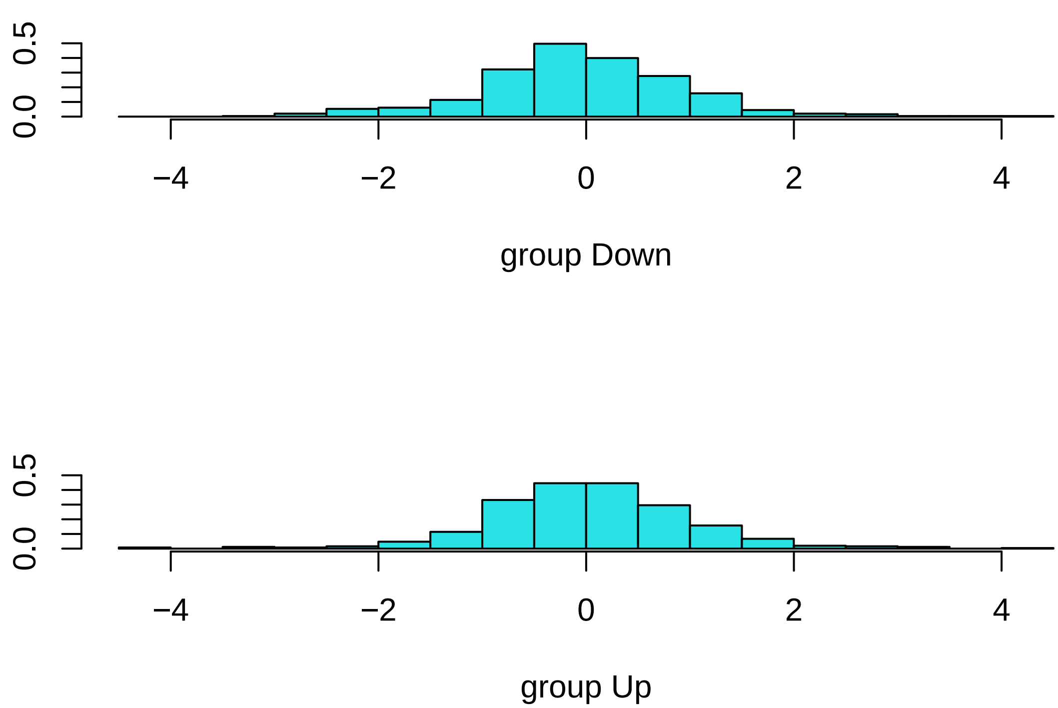
```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:ISLR2':
##
##     Boston
```

```r
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
```

```r
plot(lda.fit)
```



The LDA output indicates that $\hat{\pi}_1 = 0.492$ and $\hat{\pi}_2 = 0.508$; 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 `lagone` and `lagtwo` that are used to form the LDA decision rule. In other words, these are the multipliers of the elements of $X = x$ in (4.24). If $-0.642 \times lagone - 0.514 \times lagtwo$ 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 $-0.642 \times lagone - 0.514 \times lagtwo$ for each of the training observations. The `Up` and `Down` observations are displayed separately.

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 kth column contains the posterior probability that the corresponding observation belongs to the kth class, computed from (4.15). Finally, `x` contains the linear discriminants, described earlier.

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

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

As we observed in Section 4.5, the LDA and logistic regression predictions are almost identical.

```
lda.class <- lda.pred$class
table(lda.class, Direction.2005)
```

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

```
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] >= .5)
```

```
## [1] 70
```

```
sum(lda.pred$posterior[, 1] < .5)
```

```
## [1] 182
```

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

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

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

```r
lda.class[1:20]
```

```
##  [1] Up   Up   Up   Up   Up   Up   Up   Up   Up   Up   Up   Down Up   Up   Up  
## [16] 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%.

```r
sum(lda.pred$posterior[, 1] > .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 %.

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 syntax is identical to that of `lda()`.

```r
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.

```r
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
```

```r
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!

Naive Bayes

Next, we fit a naive Bayes model to the `Smarket` data. Naive Bayes is implemented in `R` using the `naiveBayes()` function, which is part of the `e1071` 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.

```r
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.4.1
```

```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 `lagone` 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
```

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

```
## [1] 1.227446
```

The `predict()` function is straightforward.

```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
```

```r
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. %

```r
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
```

K-Nearest Neighbors

We will now perform KNN using the `knn()` 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.

- A matrix containing the predictors associated with the training data, labeled `train.X` below.
- A matrix containing the predictors associated with the data for which we wish to make predictions, labeled `test.X` below.
- A vector containing the class labels for the training observations, labeled `train.Direction` below.
- 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 `lagone` and `lagtwo` variables together into two matrices, one for the training set and the other for the test set.

```
library(class)
```

```
## Warning: package 'class' was built under R version 4.4.1
```

```
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
```

```r
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 `Smarket` data but it does often provide impressive results. As an example we will apply the KNN approach to the `Insurance` 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 purchases a caravan insurance policy. In this data set, only 6 % of people purchased caravan insurance.

```r
dim(Caravan)
```

```
## [1] 5822   86
```

```r
attach(Caravan)
summary(Purchase)
```

```
##   No  Yes
## 5474  348
```

```r
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). As far as KNN is concerned, a difference of $\$1,000$ in salary is enormous compared to a difference of 50 years in age. Consequently, `salary` will drive the KNN classification results, and `age` will have almost no effect. This is contrary to our intuition that a salary difference of $\$1,000$ is quite small compared to an age difference of 50 years. Furthermore, the importance of scale to the KNN classifier leads to another issue: if we measured `salary` in Japanese yen, or if we measured `age` in minutes, then we'd get quite different classification results from what we get if these two variables are measured in dollars and years.

A good way to handle this problem is to *standardize* the data so that all 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.

```r
standardized.X <- scale(Caravan[, -86])
var(Caravan[, 1])
```

```
## [1] 165.0378
```

```r
var(Caravan[, 2])
```

```
## [1] 0.1647078
```

```r
var(standardized.X[, 1])
```

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

```r
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.%

```r
test <- 1:1000
train.X <- standardized.X[-test, ]
test.X <- standardized.X[test, ]
train.Y <- Purchase[-test]
test.Y <- Purchase[test]
set.seed(1)
knn.pred <- knn(train.X, test.X, train.Y, k = 1)
mean(test.Y != knn.pred)
```

```
## [1] 0.118
```

```r
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!

Suppose that there is some non-trivial cost to trying to sell insurance to a given individual. For instance, perhaps a salesperson must visit each potential customer. If the company tries to sell insurance to a random selection of customers, then the success rate will be only 6 %, which may be far too low given the costs involved. Instead, the company would like to try to sell insurance only to customers who are likely to buy it. So the overall error rate is not of interest. Instead, the fraction of individuals that are correctly predicted to buy insurance is of interest. It turns out that KNN with $K = 1$ does far better than random guessing among the customers that are predicted to buy insurance. Among 77 such customers, 9, or 11.7 %, actually do purchase insurance. This is double the rate that one would obtain from random guessing.

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

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

```r
9 / (68 + 9)
```

```
## [1] 0.1168831
```

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
```

```r
5 / 26
```

```
## [1] 0.1923077
```

```r
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
```

```r
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. Even worse, we are wrong about all of these! 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!

```r
glm.fits <- glm(Purchase ~ ., data = Caravan,
    family = binomial, subset = -test)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
glm.probs <- predict(glm.fits, Caravan[test, ],
    type = "response")
glm.pred <- rep("No", 1000)
glm.pred[glm.probs > .5] <- "Yes"
table(glm.pred, test.Y)
```

```
##         test.Y
## glm.pred  No Yes
##      No  934  59
##      Yes   7   0
```

```r
glm.pred <- rep("No", 1000)
glm.pred[glm.probs > .25] <- "Yes"
table(glm.pred, test.Y)
```

```
##         test.Y
## glm.pred  No Yes
##      No  919  48
##      Yes  22  11
```

```r
11 / (22 + 11)
```

```
## [1] 0.3333333
```

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.

```r
attach(Bikeshare)
dim(Bikeshare)
```

```
## [1] 8645   15
```

```r
names(Bikeshare)
```

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

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

```r
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
## ---
## 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.

The results seen in Section 4.6.1 used a slightly different coding of the variables `hr` and `mnth`, as follows:

```r
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
## ---
## 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:

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

```
## [1] 1.573305e-18
```

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

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

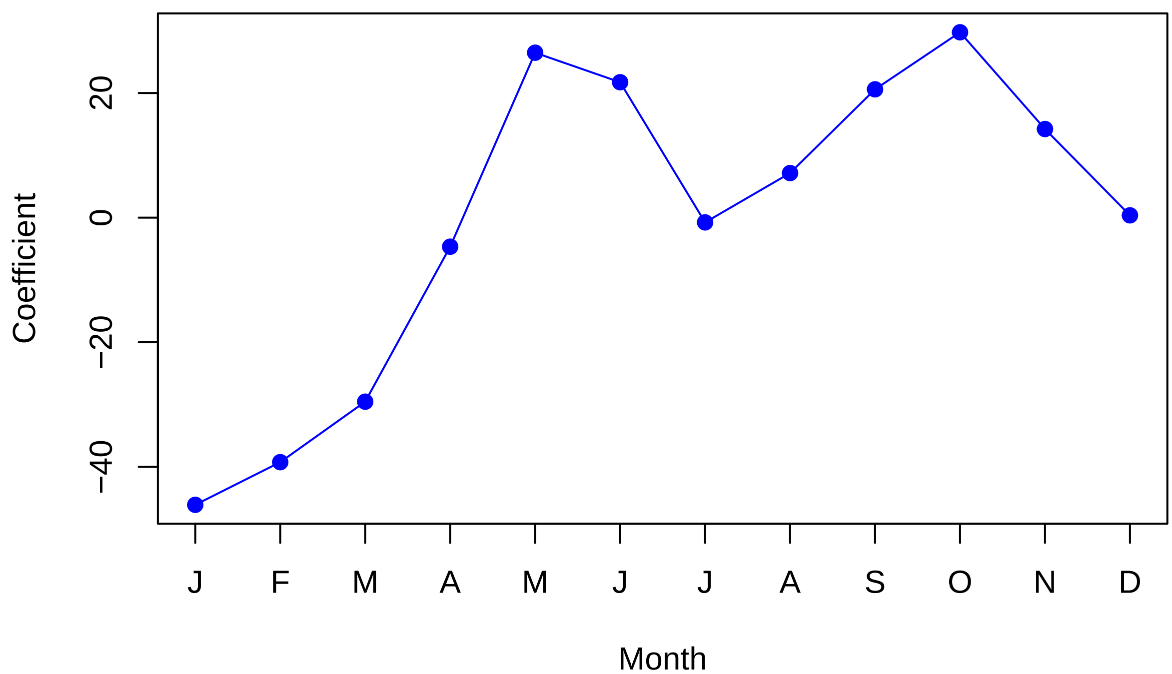
```
## [1] TRUE
```

To reproduce the left-hand side of Figure 4.13, we must first obtain the coefficient estimates associated with `mnth`. 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.

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

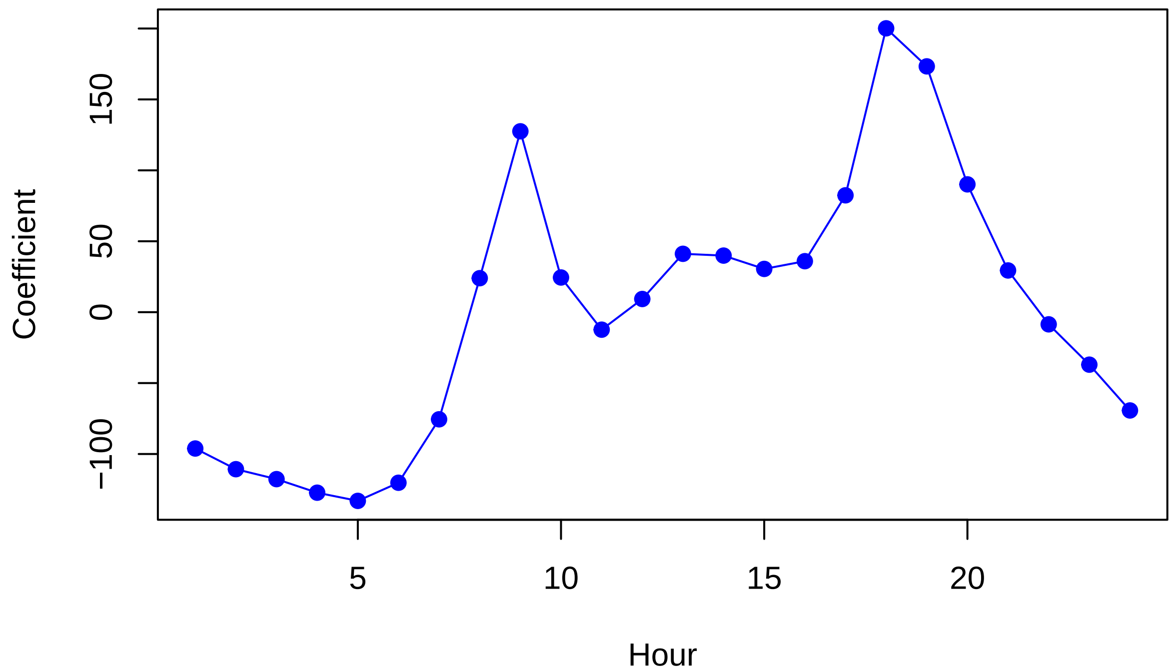
To make the plot, we manually label the x-axis with the names of the months.

```r
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.

```r
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()` with the argument `family = poisson` to specify that we wish to fit a Poisson regression model:

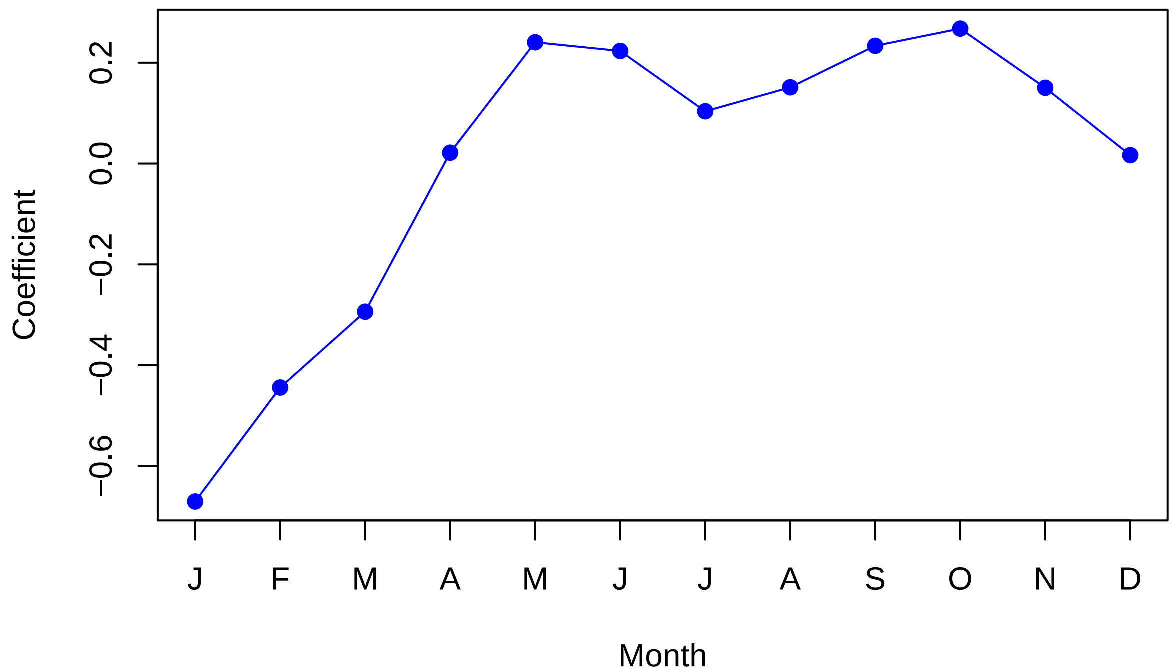
```
mod.pois <- glm(
    bikers ~ mnth + hr + workingday + temp + weathersit,
    data = Bikeshare, family = poisson
  )
summary(mod.pois)
```

```
##
## Call:
## glm(formula = bikers ~ mnth + hr + workingday + temp + weathersit,
##     family = poisson, data = Bikeshare)
##
## 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 ***
```

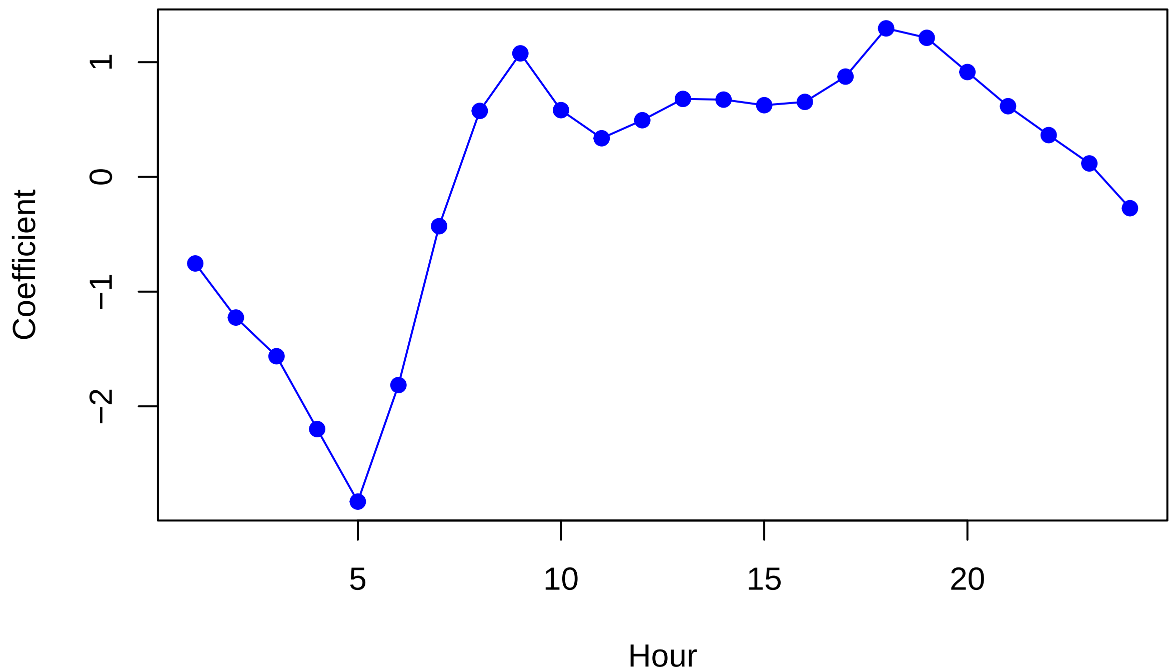
```
## mnth9                       0.233493   0.003102   75.281  < 2e-16 ***
## mnth10                      0.267573   0.002785   96.091  < 2e-16 ***
## mnth11                      0.150264   0.003180   47.248  < 2e-16 ***
## hr1                        -0.754386   0.007879  -95.744  < 2e-16 ***
## hr2                        -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 ***
## ---
## 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 Figure 4.15:

```r
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"))
```

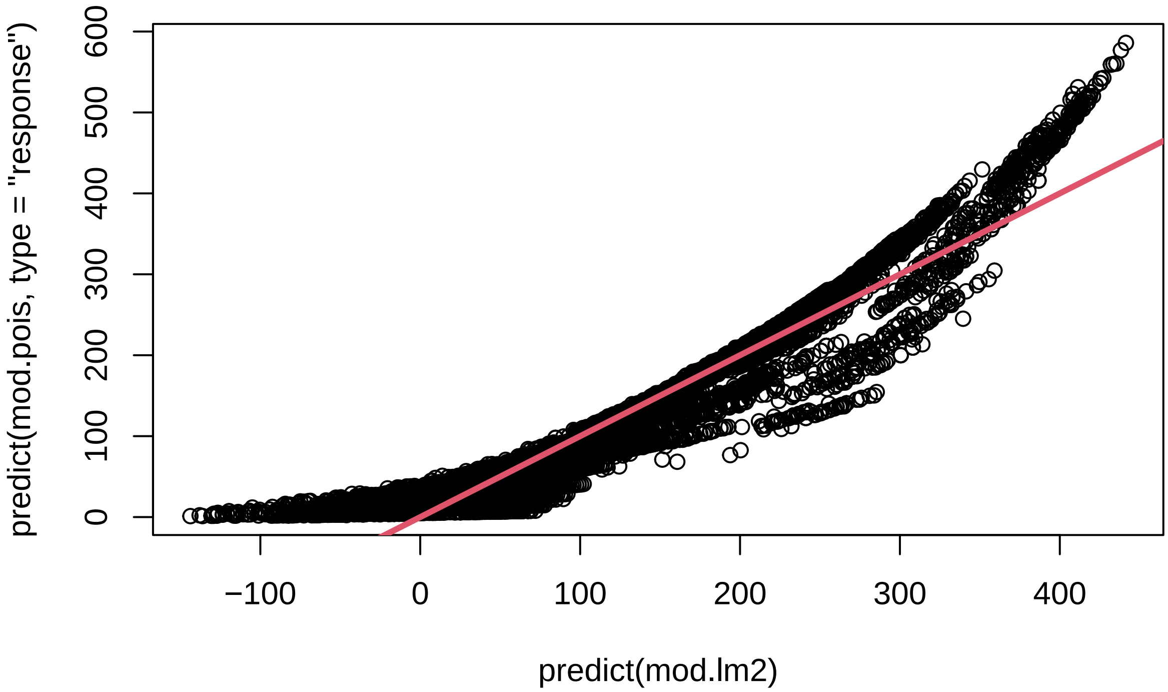



```
coef.hours <- c(coef(mod.pois)[13:35],
     -sum(coef(mod.pois)[13:35]))
plot(coef.hours, xlab = "Hour", ylab = "Coefficient",
    col = "blue", pch = 19, type = "o")
```



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 $\exp(\hat{\beta}_0 + \hat{\beta}_1 X_1 + \ldots + \hat{\beta}_p X_p)$ rather than $\hat{\beta}_0 + \hat{\beta}_1 X_1 + \ldots + \hat{\beta}_p X_p$, 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()` 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.

On Your Own

1. Suppose we collect data for a group of students in a statistics class with variables X_1 = hours studied, X_2 = undergrad GPA, and Y = receive an A. We fit a logistic regression and produce estimated coefficient, $\beta_0 = -6$, $\beta_1 = 0.05$, $\beta_2 = 1$.

(a) Estimate the probability that a student who studies for 40 h and has an undergrad GPA of 3.5 gets an A in the class.

(b) How many hours would the student in part (a) need to study to have a 50 % chance of getting an A in the class?

2. Suppose that we take a data set, divide it into equally-sized training and test sets, and then try out two different classification procedures. First we use logistic regression and get an error rate of 20 % on the training data and 30 % on the test data. Next we use 1-nearest neighbors (i.e. K = 1) and get an average error rate (averaged over both test and training data sets) of 18 %. Based on these results, which method should we prefer to use for classification of new observations? Why?

3. This question should be answered using the `Weekly` data set, which is part of the `ISLR2` package. This data is similar in nature to the `Smarket` data from this chapter's lab, except that it contains 1, 089 weekly returns for 21 years, from the beginning of 1990 to the end of 2010.

```r
library(ISLR2)
data(Weekly, package = 'ISLR2')
```

(a) Produce some numerical and graphical summaries of the Weekly data. Do there appear to be any patterns?

(b) Use the full data set to perform a logistic regression with Direction as the response and the five lag variables plus Volume as predictors. Use the summary function to print the results. Do any of the predictors appear to be statistically significant? If so, which ones?

(c) Compute the confusion matrix and overall fraction of correct predictions. Explain what the confusion matrix is telling you about the types of mistakes made by logistic regression.

(d) Now fit the logistic regression model using a training data period from 1990 to 2008, with Lag2 as the only predictor. Compute the confusion matrix and the overall fraction of correct predictions for the held out data (that is, the data from 2009 and 2010).

(e) Repeat (d) using LDA.

(f) Repeat (d) using QDA.

(g) Repeat (d) using KNN with K = 1.

(h) Repeat (d) using naive Bayes.

(i) Which of these methods appears to provide the best results on this data?

(j) Experiment with different combinations of predictors, including possible transformations and interactions, for each of the methods. Report the variables, method, and associated confusion matrix that appears to provide the best results on the held out data. Note that you should also experiment with values for K in the KNN classifier.

4. In this problem, you will develop a model to predict whether a given car gets high or low gas mileage based on the Auto data set.

```r
data(Auto, package = 'ISLR2')
```

(a) Create a binary variable, mpg01, that contains a 1 if mpg contains a value above its median, and a 0 if mpg contains a value below its median. You can compute the median using the median() function. Note you may find it helpful to use the data.frame() function to create a single data set containing both mpg01 and the other Auto variables.

(b) Explore the data graphically in order to investigate the association between mpg01 and the other features. Which of the other features seem most likely to be useful in predicting mpg01? Scatterplots and boxplots may be useful tools to answer this question. Describe your findings.

(c) Split the data into a training set and a test set.

(d) Perform LDA on the training data in order to predict mpg01 using the variables that seemed most associated with mpg01 in

(b). What is the test error of the model obtained?

(e) Perform QDA on the training data in order to predict mpg01 using the variables that seemed most associated with mpg01 in (b). What is the test error of the model obtained?

(f) Perform logistic regression on the training data in order to predict mpg01 using the variables that seemed most associated with mpg01 in (b). What is the test error of the model obtained?

(g) Perform naive Bayes on the training data in order to predict mpg01 using the variables that seemed most associated with mpg01 in (b). What is the test error of the model obtained?

(h) Perform KNN on the training data, with several values of K, in order to predict mpg01. Use only the variables that seemed most associated with mpg01 in (b). What test errors do you obtain? Which value of K seems to perform the best on this data set?