.6 Lab: Logistic Regression, LDA, QDA, and KNN

4.6.1 The Stock Market Data

We will begin by examining some numerical and graphical summaries of the `Smarket` data, which is part of the `ISLR` 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).

```
> library(ISLR)
> names(Smarket)
[1] "Year"      "Lag1"      "Lag2"      "Lag3"      "Lag4"
[6] "Lag5"      "Volume"    "Today"     "Direction"
> dim(Smarket)
[1] 1250    9
> summary(Smarket)
      Year           Lag1                Lag2
 Min.   :2001   Min.   :-4.92200   Min.   :-4.92200
 1st Qu.:2002   1st Qu.:-0.63950   1st Qu.:-0.63950
 Median :2003   Median : 0.03900   Median : 0.03900
 Mean   :2003   Mean   : 0.00383   Mean   : 0.00392
 3rd Qu.:2004   3rd Qu.: 0.59675   3rd Qu.: 0.59675
 Max.   :2005   Max.   : 5.73300   Max.   : 5.73300
      Lag3                Lag4                Lag5
 Min.   :-4.92200   Min.   :-4.92200   Min.   :-4.92200
 1st Qu.:-0.64000   1st Qu.:-0.64000   1st Qu.:-0.64000
 Median : 0.03850   Median : 0.03850   Median : 0.03850
 Mean   : 0.00172   Mean   : 0.00164   Mean   : 0.00561
 3rd Qu.: 0.59675   3rd Qu.: 0.59675   3rd Qu.: 0.59700
 Max.   : 5.73300   Max.   : 5.73300   Max.   : 5.73300
     Volume           Today           Direction
 Min.   :0.356   Min.   :-4.92200   Down:602
 1st Qu.:1.257   1st Qu.:-0.63950   Up  :648
 Median :1.423   Median : 0.03850
 Mean   :1.478   Mean   : 0.00314
 3rd Qu.:1.642   3rd Qu.: 0.59675
 Max.   :3.152   Max.   : 5.73300
> 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.

```
> cor(Smarket)
Error in cor(Smarket) : 'x' must be numeric
> cor(Smarket[,-9])
         Year     Lag1     Lag2     Lag3     Lag4     Lag5
Year   1.0000  0.02970  0.03060  0.03319  0.03569  0.02979
Lag1   0.0297  1.00000 -0.02629 -0.01080 -0.00299 -0.00567
Lag2   0.0306 -0.02629  1.00000 -0.02590 -0.01085 -0.00356
Lag3   0.0332 -0.01080 -0.02590  1.00000 -0.02405 -0.01881
Lag4   0.0357 -0.00299 -0.01085 -0.02405  1.00000 -0.02708
Lag5   0.0298 -0.00567 -0.00356 -0.01881 -0.02708  1.00000
Volume 0.5390  0.04091 -0.04338 -0.04182 -0.04841 -0.02200
Today  0.0301 -0.02616 -0.01025 -0.00245 -0.00690 -0.03486
        Volume    Today
Year    0.5390  0.03010
```

```
Lag1     0.0409 -0.02616
Lag2    -0.0434 -0.01025
Lag3    -0.0418 -0.00245
Lag4    -0.0484 -0.00690
Lag5    -0.0220 -0.03486
Volume   1.0000  0.01459
Today    0.0146  1.00000
```

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 we see that `Volume` is increasing over time. In other words, the average number of shares traded daily increased from 2001 to 2005.

```
> attach(Smarket)
> plot(Volume)
```

4.6.2 Logistic Regression

Next, we will fit a logistic regression model in order to predict `Direction` using `Lag1` through `Lag5` and `Volume`. The `glm()` function fits *generalized linear models*, a class of models that includes 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()`

generalized linear model

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

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

Deviance Residuals:
   Min      1Q  Median      3Q     Max
 -1.45   -1.20    1.07    1.15    1.33

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.12600    0.24074   -0.52     0.60
Lag1        -0.07307    0.05017   -1.46     0.15
Lag2        -0.04230    0.05009   -0.84     0.40
Lag3         0.01109    0.04994    0.22     0.82
Lag4         0.00936    0.04997    0.19     0.85
Lag5         0.01031    0.04951    0.21     0.83
Volume       0.13544    0.15836    0.86     0.39
```

```
(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: 1742

Number of Fisher Scoring iterations: 3
```

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`.

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.fit)
(Intercept)         Lag1         Lag2         Lag3         Lag4
   -0.12600     -0.07307     -0.04230      0.01109      0.00936
       Lag5       Volume
    0.01031      0.13544
> summary(glm.fit)$coef
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.12600     0.2407  -0.523    0.601
Lag1        -0.07307     0.0502  -1.457    0.145
Lag2        -0.04230     0.0501  -0.845    0.398
Lag3         0.01109     0.0499   0.222    0.824
Lag4         0.00936     0.0500   0.187    0.851
Lag5         0.01031     0.0495   0.208    0.835
Volume       0.13544     0.1584   0.855    0.392
> summary(glm.fit)$coef[,4]
(Intercept)         Lag1         Lag2         Lag3         Lag4
      0.601        0.145        0.398        0.824        0.851
       Lag5       Volume
      0.835        0.392
```

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`.

```
> glm.probs=predict(glm.fit,type="response")
> glm.probs[1:10]
    1     2     3     4     5     6     7     8     9    10
0.507 0.481 0.481 0.515 0.511 0.507 0.493 0.509 0.518 0.489
```

```
> 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.

```
> 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.

`table()`

```
> table(glm.pred,Direction)
        Direction
glm.pred Down  Up
    Down  145 141
    Up    457 507
> (507+145)/1250
[1] 0.5216
> 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.

```
> 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`. 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.

boolean

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.fit=glm(Direction~Lag1+Lag2+Lag3+Lag4+Lag5+Volume,
    data=Smarket,family=binomial,subset=train)
> glm.probs=predict(glm.fit,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(glm.pred==Direction.2005)
```

```
[1] 0.48
> mean(glm.pred!=Direction.2005)
[1] 0.52
```

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 `Lag1`. 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 `Lag1` and `Lag2`, which seemed to have the highest predictive power in the original logistic regression model.

```
> glm.fit=glm(Direction~Lag1+Lag2,data=Smarket,family=binomial,
    subset=train)
> glm.probs=predict(glm.fit,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] 0.56
> 106/(106+76)
[1] 0.582
```

Now the results appear to be more promising: 56 % of the daily movements have been correctly predicted. The confusion matrix suggests that on days when logistic regression predicts that the market will decline, it is only correct 50 % of the time. However, on days when it 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.

```
> predict(glm.fit,newdata=data.frame(Lag1=c(1.2,1.5),
    Lag2=c(1.1,-0.8)),type="response")
         1          2
    0.4791     0.4961
```

4.6.3 Linear Discriminant Analysis

Now we will perform LDA on the `Smarket` data. In `R`, we fit a 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. `lda()`

```
> 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
0.492 0.508

Group means:
        Lag1    Lag2
Down  0.0428  0.0339
Up   -0.0395 -0.0313

Coefficients of linear discriminants:
        LD1
Lag1 -0.642
Lag2 -0.514
> 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 `Lag1` and `Lag2` that are used to form the LDA decision rule. In other words, these are the multipliers of the elements of $X = x$ in (4.19). 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 `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 (4.10). 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.pred Down  Up
    Down   35  35
    Up     76 106
> mean(lda.class==Direction.2005)
[1] 0.56
```

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*:

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

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]>.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 %.

4.6.4 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()`. `qda()`

```
> 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
0.492 0.508

Group means:
        Lag1    Lag2
Down  0.0428  0.0339
Up   -0.0395 -0.0313
```

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] 0.599
```

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!

4.6.5 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. `knn()`

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.