Classification

We will use the `Smarket` data. 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.

```
library(MASS)
library(ISLR2)
```

```
##
##    'ISLR2'

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

```
attach(Smarket)
```

Logistic Regression

We will fit a logistic regression model in order to predict `direction` using `lagone` through `lagfive` and `volume`. 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.

```
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)
##
## 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
```

The `predict()` function can be used to predict the probability that the market will go up, given values of the predictors. If no data set is supplied to the `predict()` function, then the probabilities are computed for the training data. 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.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
```

```
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"
```

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

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

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

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

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

$100\% - 52.2\% = 47.8\%$, is the training error rate. To estimate the test error rate, we can fit the model using part of the data, and then examine how well it predicts the held out data. 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]
```

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.

```
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")
```

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] 0.4801587
```

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

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

The test error rate estimate is 52,%. 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). 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.

```
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
```

```r
106 / (106 + 76)
```

```
## [1] 0.5824176
```

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.

```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
```

Linear Discriminant Analysis

We fit an LDA model using the `lda()` function, which is part of the `MASS` library.

```r
lda.fit <- lda(Direction ~ Lag1 + Lag2, data = Smarket,
    subset = train)
lda.pred <- predict(lda.fit, Smarket.2005)
```

The LDA and logistic regression predictions are almost identical.

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

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

```r
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`.

```r
sum(lda.pred$posterior[, 1] >= .5)
```

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

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

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

Notice that the posterior probability `lda.pred$posterior[, 1]` 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
```

```
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
```

Use a posterior probability threshold other than 50,% to make predictions:

```
sum(lda.pred$posterior[, 1] > .4)
```

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

Quadratic Discriminant Analysis

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

Naive Bayes

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()`.

```
library(e1071)
nb.fit <- naiveBayes(Direction ~ Lag1 + Lag2, data = Smarket,
    subset = train)
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
```

K-Nearest Neighbors

We will now perform KNN using the `knn()` function, which is part of the `class` library. `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 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)
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
```

```
mean(knn.pred == Direction.2005)
```

```
## [1] 0.5
```

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.