

Classification

James D. Wilson

March 23rd, 2018

In this presentation, we will investigate how to use the following classification methods:

- 1) Linear Discriminant Analysis
- 2) Quadratic Discriminant Analysis
- 3) K-Nearest Neighbors

Much of this lab is a replication of Section 4.6 in the “Introduction to Statistical Learning” textbook by James, Witten, Hastie, and Tibshirani (with my own commentary throughout).

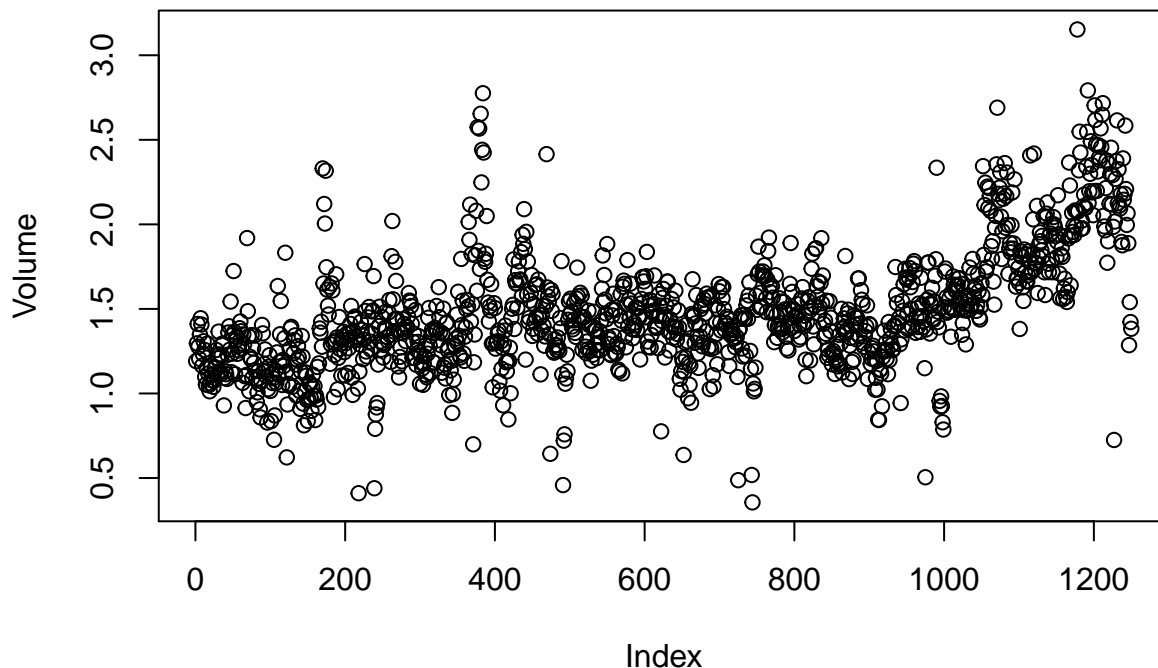
We’ll be analyzing the Stock Market Data, *Smarket*, in the *ISLR* library. Let’s download the data and look at a review.

```
library(ISLR, quietly = TRUE)
?Smarket
```

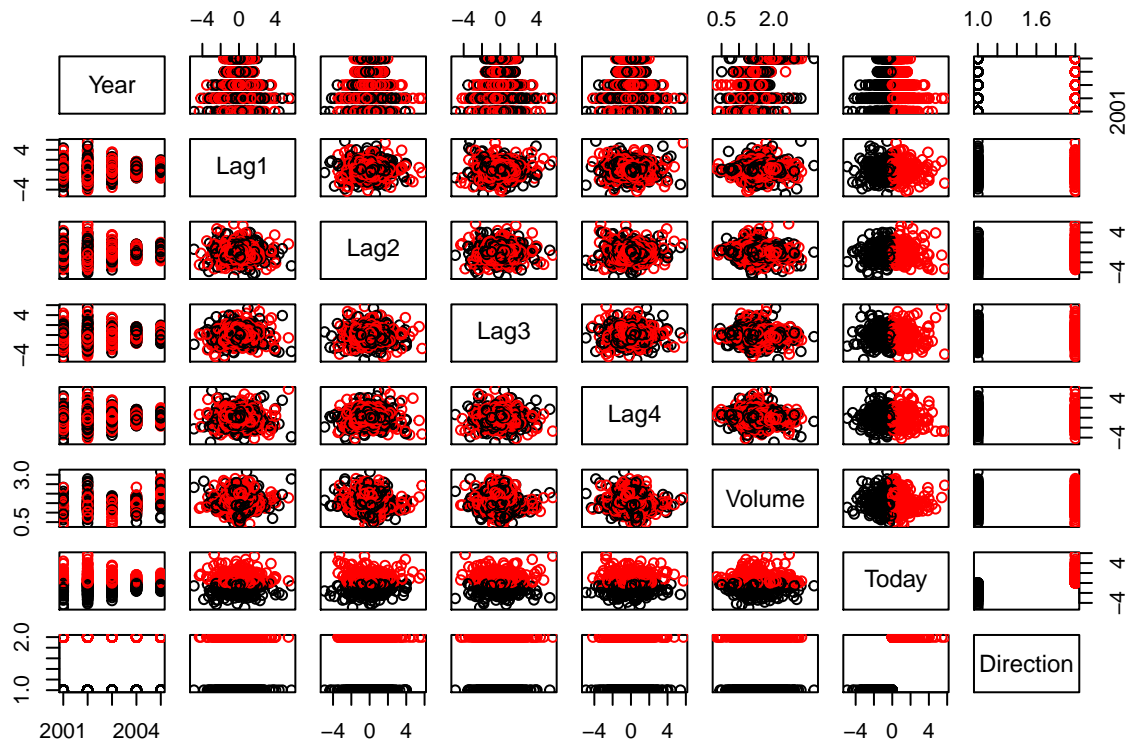
The goal will be to predict the *Direction* of the stock market using the percentage returns *Lag1* - *Lag5*, and the *Volume* of shares traded the previous day.

Let’s explore a bit of the data to get an idea of the relationships between the covariates and the *Direction*. We note that there is also a variable *Today* in the dataset which directly quantifies the percentatge return on the date in question. (This variable will perfectly capture the *Direction*)

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



```
pairs(Smarket[, -6], col = Smarket$Direction)
```



First, let's separate the data into a training and test set for further evaluation. In particular, we will train on the data for years before 2005 and then evaluate our predictor on the test set containing data from the year 2005.

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

Linear Discriminant Analysis

We will now implement Linear Discriminant Analysis on the *Smarket* data. To do so, we need to first install the *MASS* library in R.

```
install.packages("MASS", repos = 'http://cran.us.r-project.org')
```

```
##
## The downloaded binary packages are in
## /var/folders/kw/bhptl1817xgd3tpk4xl94h9w0000xb/T/Rtmpj1mBAZ/downloaded_packages
library(MASS, quietly = TRUE)
```

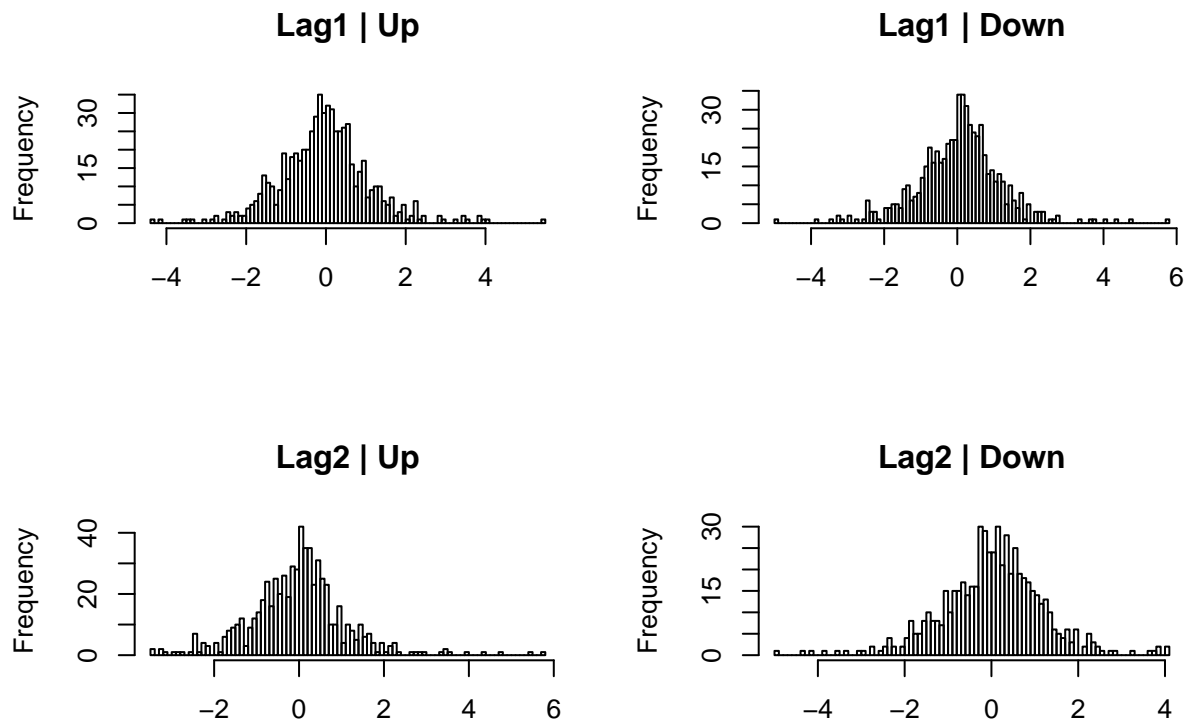
To implement LDA, we simply use the *lda()* function in a similar fashion as linear and logistic regression. Below, we fit an LDA to the training data in the years 2000 - 2004 and then classify the market *Direction* in the year 2005.

Here, we will only use the variables *Lag1* and *Lag2* since they had the best performance in logistic regression. In general, however, we'd want to check various combinations of the predictors to determine which gives the best classifier.

First, we preview the class conditional distributions to see if the Normal assumptions are reasonable.

```
par(mfrow = c(2,2))
hist(Lag1[Direction == "Up"], n = 100, main = "Lag1 | Up", xlab = "")
```

```
hist(Lag1[Direction == "Down"], n = 100, main = "Lag1 | Down", xlab = "")
hist(Lag2[Direction == "Up"], n = 100, main = "Lag2 | Up", xlab = "")
hist(Lag2[Direction == "Down"], n = 100, main = "Lag2 | Down", xlab = "")
```



The Normal assumptions seem reasonable. Of course, we could dig deeper to check these assumptions using your favorite test for normality. But, we don't worry about that in this case.

Now we perform LDA.

```
#run LDA on the training data
lda.fit <- lda(Direction ~ Lag1 + Lag2, data = Smarket, subset = train)

#look at a summary of the fit
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
```

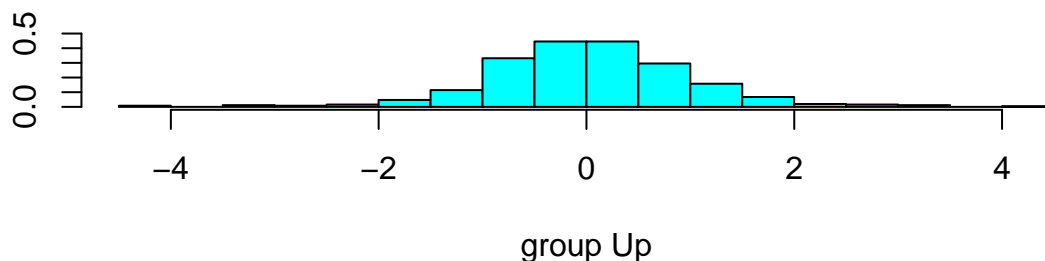
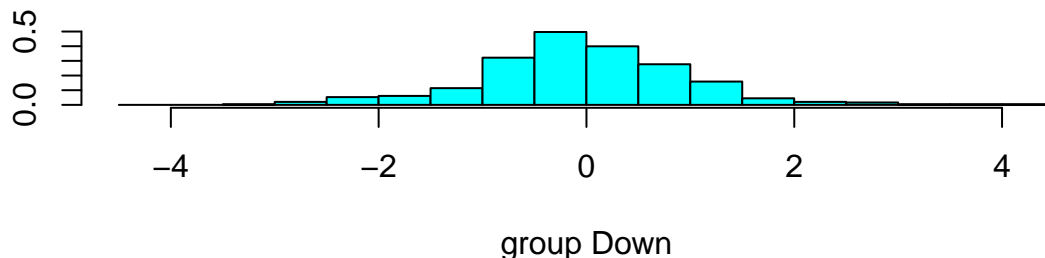
Above, the coefficients of the linear discriminants give the coefficients associated with each of the predictors that dictate the linear discriminants. In this situation, if the value

$$-0.642 * Lag1 - 0.514 * Lag2$$

is large, then the LDA classifier will predict a market increase. If this value is small, then it will predict a market decrease.

We can plot the discriminants on the training data

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



Next, we predict the market value in 2005 using our LDA classification rule.

```
#predict market Direction in 2005.
```

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

```
names(lda.pred)
```

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

```
#Look at the confusion matrix to evaluate how the method performed
```

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

```
table(lda.class, Direction.2005)
```

```
##           Direction.2005
```

```
## lda.class Down Up
```

```
##      Down   35  35
```

```
##      Up    76 106
```

```
#calculate the accuracy
```

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

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

```
#the posterior probability that the market will decrease
#let's look at the first 20 and compare with our guess
```

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

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

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

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

It's interesting to note that the greatest posterior probability of decrease in all of 2005 was 52.02%. This gives us an idea of how challenging it is to classify the market value.

Quadratic Discriminant Analysis

Running quadratic discriminant analysis (QDA) is analagous to LDA as implemented above. As a result, I will just run the analysis with little commentary as the functions are all the same as above.

```
#run QDA on the training data
```

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

```
#look at summary of fit
```

```
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
```

```
#predict the market Direction in 2005
```

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

```
#look at the confusion matrix
```

```
table(qda.class, Direction.2005)
```

```
##           Direction.2005
```

```
## qda.class Down  Up
```

```
##      Down   30  20
```

```
##      Up    81 121
```

```
#calculate the accuracy
```

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

```
## [1] 0.5992063
```

QDA has an accuracy of nearly 60% for the market direction in 2005. So far this is the best result, suggesting that a quadratic discriminant outperforms the linear discriminants calculated by LDA and logistic regression.

K-Nearest Neighbors

We will now implement the K-Nearest Neighbors algorithm on the *Smarket* data set and compare with the results of the methods above. To do this, we use the *knn()* function from the *class* library. Let's first install the *class* library.

```
install.packages("class", repos = 'http://cran.us.r-project.org')

##
## The downloaded binary packages are in
## /var/folders/kw/bhptl1817xgd3tpk4xl94h9w0000xb/T//Rtmpj1mBAZ/downloaded_packages
library(class, quietly = TRUE)

?knn
```

The *knn()* function requires 4 different inputs as follows:

- 1) *train* : a matrix containing the predictors for the training data
- 2) *test* : a matrix containing the predictors for the test data
- 3) *cl*: the labels (classes) of the training data
- 4) *k*: the number of nearest neighbors you'd like to use

We now prepare our data and run *knn* using $K = 1$ and $K = 3$ for demonstration. In practice, we would want to run *knn* across many values of K and choose the K that works best. This can be done via cross-validation using the *knn.cv()* function, but I don't do this below. The arguments for this function are exactly the same as those in the *knn* function.

```
#prepare the data
train.X <- cbind(Lag1, Lag2)[train, ]
test.X <- cbind(Lag1, Lag2)[!train, ]
train.Direction <- Direction[train]

#run KNN with K = 1
set.seed(1) #for reproducibility for the randomly breaking of ties
knn.pred <- knn(train.X, test.X, train.Direction, k = 1)

#look at the confusion matrix
table(knn.pred, Direction.2005)

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

#calculate the accuracy
mean(knn.pred == Direction.2005)

## [1] 0.5
```

Using KNN with $K = 1$ is exactly the same as randomly guessing. So, we now repeat the above using $K = 3$.

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

```
#look at the confusion matrix  
table(knn.pred, Direction.2005)
```

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

```
#calculate the accuracy  
mean(knn.pred == Direction.2005)
```

```
## [1] 0.5357143
```

I invite you to consider repeating the above exercise using cross-validation to choose the best K on the training data.