Lab 4 - Logistic Regression, LDA, QDA and KNN

An Introduction to Statistical Learning

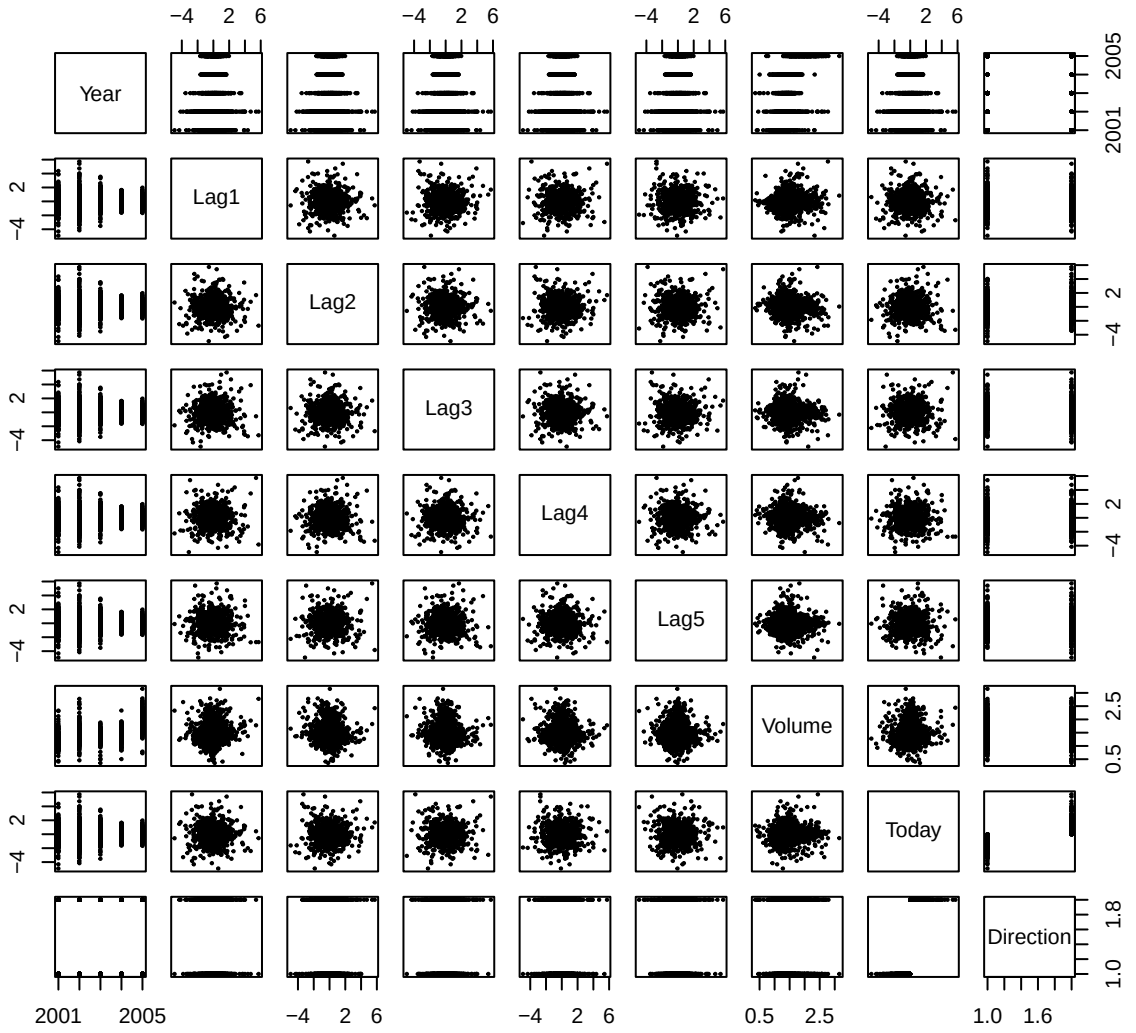
We are going to work with the `Smarket` dataset:

```
library(ISLR)
attach(Smarket)
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
##
##
##
##
```

Let's plot the data:

```
pairs(Smarket, cex=.25)
```



We will try to predict the value for `Direction`, that takes two values:

```
contrasts(Direction)
```

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

1. Logistic Regression

Creating train and a test sets

All samples whose `Year` value is less than 2005 will be used to train the model:

```
train = (Year<2005)
Smarket.train = Smarket[train,]
```

We generate the test set in the same manner, and the test target vector:

```
Smarket.test = Smarket[!train,]
Direction.test = Direction[!train]
```

Fitting the model

Let's fit a *generalized linear model* (`glm`) for logistic regression, using the training set.

To specify the training subset we could do:

- Using `data=Smarket.train`
- Using `data=Smarket, subset=train`

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

The parameter `family='binomial'` tells `R` to perform a logistic regression instead of other type of linear model.

```
summary(glm.fit)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
##     Volume, family = "binomial", data = Smarket, subset = train)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.302  -1.190   1.079   1.160   1.350
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.191213   0.333690   0.573    0.567
## Lag1        -0.054178   0.051785  -1.046    0.295
## Lag2        -0.045805   0.051797  -0.884    0.377
## Lag3         0.007200   0.051644   0.139    0.889
## Lag4         0.006441   0.051706   0.125    0.901
## Lag5        -0.004223   0.051138  -0.083    0.934
## Volume      -0.116257   0.239618  -0.485    0.628
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1383.3  on 997  degrees of freedom
## Residual deviance: 1381.1  on 991  degrees of freedom
## AIC: 1395.1
##
## Number of Fisher Scoring iterations: 3
```

From the summary we find out that the predictor with the smallest p-value is `Lag1`, but it's quite large, 0.145, so its significance is relative.

We can plot the p-values from the `summary`:

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

```
## (Intercept)        Lag1        Lag2        Lag3        Lag4        Lag5
##   0.5666278   0.2954646   0.3765201   0.8891200   0.9008654   0.9341907
##      Volume
##   0.6275518
```

From now on we will use a model with only two predictors: `Lag1` and `Lag2`:

```
glm.fit = glm(Direction ~ Lag1+Lag2, data=Smarket, subset=train, family='binomial')
summary(glm.fit)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2, family = "binomial", data = Smarket,
##     subset = train)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.345  -1.188   1.074   1.164   1.326
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.03222    0.06338   0.508    0.611
## Lag1        -0.05562    0.05171  -1.076    0.282
## Lag2        -0.04449    0.05166  -0.861    0.389
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1383.3  on 997  degrees of freedom
## Residual deviance: 1381.4  on 995  degrees of freedom
## AIC: 1387.4
##
## Number of Fisher Scoring iterations: 3
```

Predicting `Direction`

In this case we will make predictions using the same samples that were used to fit the model. I we were to use a test subset we would pass the parameter `newdata=test_data`, where `test_data` is a subset of the data that has not been used to train the model.

We will use the test set to make predictions; to specify the test set we pass the parameter `newdata=Smarket.test`, where `Smarket.test` is the previously generated subset of the data that has not been used to train the model.

We are doing a logistic regression, so we need to get predictions in the form of probabilities of belonging to a class, or $p(Y = 1 \mid X)$. To do so, we need to specify the parameter `type='response'`:

```
glm.probs = predict(glm.fit, newdata=Smarket.test, type='response')
glm.probs[1:8]
```

```
##       999      1000      1001      1002      1003      1004      1005      1006
## 0.5098275 0.5208237 0.5332635 0.5260574 0.5072103 0.5061388 0.5048890 0.5127302
```

Predicted probabilities greater than 1 assign the sample to the class `Up`, as seen with the `contrasts` function.

At this point we have predictions in the form of probabilities, but we need to have predictions in the form `Up` or `Down`.

```
glm.preds = rep('Down', dim(Smarket.test)[1])
glm.preds[glm.probs > 0.5] = 'Up'
glm.preds[1:8]
```

```
## [1] "Up" "Up" "Up" "Up" "Up" "Up" "Up" "Up"
```

Checking the model performance

We produce a confussion matrix:

```
table(glm.preds, Direction.test)
```

```
##          Direction.test
## glm.preds Down  Up
##      Down   35  35
##      Up     76 106
```

To compute the ratio of correctly predicted values:

```
(35+106)/dim(Smarket.test)[1]
```

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

or:

```
mean(glm.preds == Direction.test)
```

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

We find that the model gave the correct answer for the 56% fo the test data

Predicting values for particular predictor values

If we want to predict the output for a set of known predictor inputs:

```
glm.preds.2 = predict(glm.fit, type='response',
                      newdata = data.frame(Lag1=c(1.2, 1.5), Lag2=c(1.1, -0.8)))
```

Here we give two new "samples", with `(Lag1, Lag2)` equal to $(1.2, 1.1)$ and $(1.5, -0.8)$.

The predicted outputs for each of the two samples are:

```
glm.preds.2
```

```
##         1         2
## 0.4791462 0.4960939
```

2. Linear Discriminant Analysis (LDA)

We need to load the `MASS` library to use `LDA` and `QDA`.

```
library(MASS)
```

Fitting the model

We will use the same train and test sets from the Logistic Regression model.

```
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
```

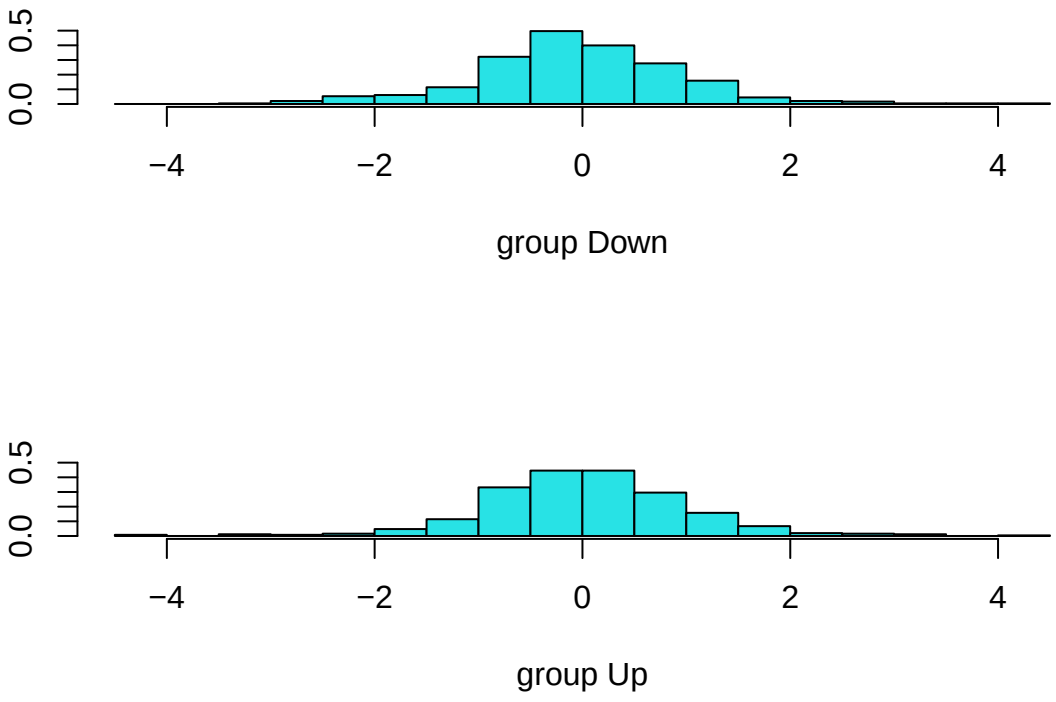
The coefficients of the linear discriminants are the values that provide the linear combination of `Lag1` and `Lag2` used to form the LDA decision rule (the multipliers of the elements $X = x$ in equation 4.19 of the book):

$$-0.642 \times \text{Lag1} - 0.513 \times \text{Lag2}$$

When this value is large then the LDA classifier will predict a market increase.

We can plot the linear discriminants, obtained by computing the previous linear combination for each of the training samples:

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



We also get two **group means**: the average of each predictor within each class, which are used by `LDA` as estimates of μ_k in equation 4.19 of the book.

Making predictions

```
lda.pred = predict(lda.fit, newdata = Smarket.test)
```

The obtained predictions have the following components:

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

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

`class` contains the actual class labels for each prediction:

```
lda.pred$class[1:6]
```

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

`posterior` is a matrix whose kth column contains the posterior probability that the corresponding observation belongs to the kth class:

```
lda.pred$posterior[1:6,]
```

```
##           Down        Up
## 999  0.4901792 0.5098208
## 1000 0.4792185 0.5207815
## 1001 0.4668185 0.5331815
## 1002 0.4740011 0.5259989
## 1003 0.4927877 0.5072123
## 1004 0.4938562 0.5061438
```

`x` contains the scores of test cases on the discriminant variables:

```
lda.pred$x[1:6,]
```

```
##         999        1000        1001        1002        1003        1004
##  0.08293096  0.59114102  1.16723063  0.83335022 -0.03792892 -0.08743142
```

Model performance

We need to compare the predicted `class` with the correct class in the test target vector:

```
table(lda.pred$class, Direction.test)
```

```
##       Direction.test
##        Down  Up
##   Down   35  35
##   Up     76 106
```

```
mean(lda.pred$class == Direction.test)
```

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

We have predicted the correct class for the 56% of the test data, exactly the same result than with Logistic Regression. In fact, both confussion matrices are identical.

Using a different probability threshold

`lda` applies a probability threshold of 0.5 when making predictions. If we want to use a different value:

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

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

```
sum(lda.pred$posterior[,1] >= 0.45)
```

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

A note on the posterior probabilities

The posterior probability corresponds to the probability that the market will *decrease*:

```
lda.pred$posterior[1:15]
```

```
##  [1] 0.4901792 0.4792185 0.4668185 0.4740011 0.4927877 0.4938562 0.4951016
##  [8] 0.4872861 0.4907013 0.4844026 0.4906963 0.5119988 0.4895152 0.4706761
## [15] 0.4744593
```

```
lda.pred$class[1:15]
```

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

When the posterior is **greater than** 0.5 the predicted class is the first class, the one corresponding to 0, i.e. `Down`.

```
contrasts(Direction.test)
```

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

3. Quadratic Discriminant Analysis (QDA)

The process is very similar than the one for LDA:

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

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

This time the model doesn't contain the coefficiens of the linear discriminants, because QDA involve quadratic functions of the predictors.

Making predictions

```
qda.pred = predict(qda.fit, newdata = Smarket.test)
```

Model performance

```
table(qda.pred$class, Direction.test)
```

```
##       Direction.test
##        Down  Up
##   Down   30  20
##   Up     81 121
```

```
mean(qda.pred$class == Direction.test)
```

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

With QDA we get 60% of the samples correctly labeled.

4. K-Nearest Neighbors (KNN)

K-Nearest Neighbors is done using the `knn` function from the `class` library.

```
library(class)
```

`knn` has a slightly different format for the input data. It takes two matrices containing the train and test data, so we need to create them using `cbind`:

```
x.train = cbind(Lag1, Lag2)[train,]
x.test = cbind(Lag1, Lag2)[!train,]
y.train = Direction[train]
```

We need to be sure that `R` will be consistently making groups through the tests, when some observations are tied as nearest neighbors.

```
set.seed(1)
```

We will try to predict outputs based only on the nearest observation (k = 1)

This method has not a fit-predict cycle. Predictions are directly made from the training data.

```
knn.pred = knn(train = x.train, test = x.test, cl = y.train, k = 1)
```

`cl` is the factor of true classifications of the training set (the target), and `k` is the number of clusters to create.

We check the model performance:

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

```
##         Direction.test
## knn.pred Down Up
##     Down   43 58
##     Up     68 83
```

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

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

In this case, 50% of the samples are correctly labeled.

We now try with k = 3:

```
knn.pred = knn(train = x.train, test = x.test, cl = y.train, k = 3)
table(knn.pred, Direction.test)
```

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

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

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

Now 53% of the observations are correctly labeled.

A note on predictor scales

KNN works with *distances* between observations, and therefore the scale of the variables influences the result; variables that are on a large scale will have a much larger effect on the distance between observations.

In the previous example, both `Lag1` and `Lag2` have the same meaning and scale, so it's not necessary to rescale them, but when working with variables that have different units or ranges we must **standardize** them using `scale`:

```
x1 = rnorm(100, 2, 5)
summary(x1)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -6.401  -2.334   1.462   1.949   5.477  15.293
```

```
x1.standardized = scale(x1)
summary(c(x1.standardized)) # c() only converts the column to a vector for visualization
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -1.70260 -0.87335 -0.09945  0.00000  0.71918  2.72073
```