Clasificación R

Juanjo Sierra

4 de diciembre de 2018

Clasificación en R

Vamos a utilizar el dataset "Breast Cancer Wisconsin Diagnostic", que cuenta con 569 ejemplos de 32 características cada uno, identificando cada uno como cáncer benigno (B) o maligno (M).

```
wbcd = read.csv("Datos/wisc_bc_data.csv", stringsAsFactors = FALSE)
str(wbcd)
```

```
## 'data.frame':    569 obs. of  32 variables:
##  $ id               : int  87139402 8910251 905520 868871 9012568 906539 925291 87880 862989 89827 .
##  $ diagnosis        : chr  "B" "B" "B" "B" ...
##  $ radius_mean      : num  12.3 10.6 11 11.3 15.2 ...
##  $ texture_mean     : num  12.4 18.9 16.8 13.4 13.2 ...
##  $ perimeter_mean   : num  78.8 69.3 70.9 73 97.7 ...
##  $ area_mean        : num  464 346 373 385 712 ...
##  $ smoothness_mean  : num  0.1028 0.0969 0.1077 0.1164 0.0796 ...
##  $ compactness_mean : num  0.0698 0.1147 0.078 0.1136 0.0693 ...
##  $ concavity_mean   : num  0.0399 0.0639 0.0305 0.0464 0.0339 ...
##  $ points_mean      : num  0.037 0.0264 0.0248 0.048 0.0266 ...
##  $ symmetry_mean    : num  0.196 0.192 0.171 0.177 0.172 ...
##  $ dimension_mean   : num  0.0595 0.0649 0.0634 0.0607 0.0554 ...
##  $ radius_se        : num  0.236 0.451 0.197 0.338 0.178 ...
##  $ texture_se       : num  0.666 1.197 1.387 1.343 0.412 ...
##  $ perimeter_se     : num  1.67 3.43 1.34 1.85 1.34 ...
##  $ area_se          : num  17.4 27.1 13.5 26.3 17.7 ...
##  $ smoothness_se    : num  0.00805 0.00747 0.00516 0.01127 0.00501 ...
##  $ compactness_se   : num  0.0118 0.03581 0.00936 0.03498 0.01485 ...
##  $ concavity_se     : num  0.0168 0.0335 0.0106 0.0219 0.0155 ...
##  $ points_se        : num  0.01241 0.01365 0.00748 0.01965 0.00915 ...
##  $ symmetry_se      : num  0.0192 0.035 0.0172 0.0158 0.0165 ...
##  $ dimension_se     : num  0.00225 0.00332 0.0022 0.00344 0.00177 ...
##  $ radius_worst     : num  13.5 11.9 12.4 11.9 16.2 ...
##  $ texture_worst    : num  15.6 22.9 26.4 15.8 15.7 ...
##  $ perimeter_worst  : num  87 78.3 79.9 76.5 104.5 ...
##  $ area_worst       : num  549 425 471 434 819 ...
##  $ smoothness_worst : num  0.139 0.121 0.137 0.137 0.113 ...
##  $ compactness_worst: num  0.127 0.252 0.148 0.182 0.174 ...
##  $ concavity_worst  : num  0.1242 0.1916 0.1067 0.0867 0.1362 ...
##  $ points_worst     : num  0.0939 0.0793 0.0743 0.0861 0.0818 ...
##  $ symmetry_worst   : num  0.283 0.294 0.3 0.21 0.249 ...
##  $ dimension_worst  : num  0.0677 0.0759 0.0788 0.0678 0.0677 ...
```

```
head(wbcd)
```

```
##         id diagnosis radius_mean texture_mean perimeter_mean area_mean
## 1 87139402         B       12.32        12.39          78.85     464.1
## 2  8910251         B       10.60        18.95          69.28     346.4
## 3   905520         B       11.04        16.83          70.92     373.2
```

```
## 4   868871         B       11.28        13.39          73.00     384.8
## 5  9012568         B       15.19        13.21          97.65     711.8
## 6   906539         B       11.57        19.04          74.20     409.7
##   smoothness_mean compactness_mean concavity_mean points_mean
## 1         0.10280          0.06981        0.03987     0.03700
## 2         0.09688          0.11470        0.06387     0.02642
## 3         0.10770          0.07804        0.03046     0.02480
## 4         0.11640          0.11360        0.04635     0.04796
## 5         0.07963          0.06934        0.03393     0.02657
## 6         0.08546          0.07722        0.05485     0.01428
##   symmetry_mean dimension_mean radius_se texture_se perimeter_se area_se
## 1        0.1959        0.05955    0.2360     0.6656        1.670   17.43
## 2        0.1922        0.06491    0.4505     1.1970        3.430   27.10
## 3        0.1714        0.06340    0.1967     1.3870        1.342   13.54
## 4        0.1771        0.06072    0.3384     1.3430        1.851   26.33
## 5        0.1721        0.05544    0.1783     0.4125        1.338   17.72
## 6        0.2031        0.06267    0.2864     1.4400        2.206   20.30
##   smoothness_se compactness_se concavity_se points_se symmetry_se
## 1      0.008045       0.011800      0.01683  0.012410     0.01924
## 2      0.007470       0.035810      0.03354  0.013650     0.03504
## 3      0.005158       0.009355      0.01056  0.007483     0.01718
## 4      0.011270       0.034980      0.02187  0.019650     0.01580
## 5      0.005012       0.014850      0.01551  0.009155     0.01647
## 6      0.007278       0.020470      0.04447  0.008799     0.01868
##   dimension_se radius_worst texture_worst perimeter_worst area_worst
## 1     0.002248        13.50         15.64           86.97      549.1
## 2     0.003318        11.88         22.94           78.28      424.8
## 3     0.002198        12.41         26.44           79.93      471.4
## 4     0.003442        11.92         15.77           76.53      434.0
## 5     0.001767        16.20         15.73          104.50      819.1
## 6     0.003339        13.07         26.98           86.43      520.5
##   smoothness_worst compactness_worst concavity_worst points_worst
## 1           0.1385            0.1266         0.12420      0.09391
## 2           0.1213            0.2515         0.19160      0.07926
## 3           0.1369            0.1482         0.10670      0.07431
## 4           0.1367            0.1822         0.08669      0.08611
## 5           0.1126            0.1737         0.13620      0.08178
## 6           0.1249            0.1937         0.25600      0.06664
##   symmetry_worst dimension_worst
## 1         0.2827         0.06771
## 2         0.2940         0.07587
## 3         0.2998         0.07881
## 4         0.2102         0.06784
## 5         0.2487         0.06766
## 6         0.3035         0.08284
```

Preprocesamiento

En primer lugar se eliminará la característica `id` que no aporta ninguna información en este caso.

```
wbcd = wbcd[,-1]
head(wbcd)
```

```
##   diagnosis radius_mean texture_mean perimeter_mean area_mean
```

```
## 1         B       12.32        12.39          78.85     464.1
## 2         B       10.60        18.95          69.28     346.4
## 3         B       11.04        16.83          70.92     373.2
## 4         B       11.28        13.39          73.00     384.8
## 5         B       15.19        13.21          97.65     711.8
## 6         B       11.57        19.04          74.20     409.7
##   smoothness_mean compactness_mean concavity_mean points_mean
## 1         0.10280          0.06981        0.03987     0.03700
## 2         0.09688          0.11470        0.06387     0.02642
## 3         0.10770          0.07804        0.03046     0.02480
## 4         0.11640          0.11360        0.04635     0.04796
## 5         0.07963          0.06934        0.03393     0.02657
## 6         0.08546          0.07722        0.05485     0.01428
##   symmetry_mean dimension_mean radius_se texture_se perimeter_se area_se
## 1        0.1959        0.05955    0.2360     0.6656        1.670   17.43
## 2        0.1922        0.06491    0.4505     1.1970        3.430   27.10
## 3        0.1714        0.06340    0.1967     1.3870        1.342   13.54
## 4        0.1771        0.06072    0.3384     1.3430        1.851   26.33
## 5        0.1721        0.05544    0.1783     0.4125        1.338   17.72
## 6        0.2031        0.06267    0.2864     1.4400        2.206   20.30
##   smoothness_se compactness_se concavity_se points_se symmetry_se
## 1      0.008045       0.011800      0.01683  0.012410     0.01924
## 2      0.007470       0.035810      0.03354  0.013650     0.03504
## 3      0.005158       0.009355      0.01056  0.007483     0.01718
## 4      0.011270       0.034980      0.02187  0.019650     0.01580
## 5      0.005012       0.014850      0.01551  0.009155     0.01647
## 6      0.007278       0.020470      0.04447  0.008799     0.01868
##   dimension_se radius_worst texture_worst perimeter_worst area_worst
## 1     0.002248        13.50         15.64           86.97      549.1
## 2     0.003318        11.88         22.94           78.28      424.8
## 3     0.002198        12.41         26.44           79.93      471.4
## 4     0.003442        11.92         15.77           76.53      434.0
## 5     0.001767        16.20         15.73          104.50      819.1
## 6     0.003339        13.07         26.98           86.43      520.5
##   smoothness_worst compactness_worst concavity_worst points_worst
## 1           0.1385            0.1266         0.12420      0.09391
## 2           0.1213            0.2515         0.19160      0.07926
## 3           0.1369            0.1482         0.10670      0.07431
## 4           0.1367            0.1822         0.08669      0.08611
## 5           0.1126            0.1737         0.13620      0.08178
## 6           0.1249            0.1937         0.25600      0.06664
##   symmetry_worst dimension_worst
## 1         0.2827         0.06771
## 2         0.2940         0.07587
## 3         0.2998         0.07881
## 4         0.2102         0.06784
## 5         0.2487         0.06766
## 6         0.3035         0.08284
```

Comprobamos cómo se distribuyen los ejemplos en base a los valores de `diagnosis`, la variable a predecir.

```
table(wbcd$diagnosis)
```

```
##
##   B   M
```

```
## 357 212
```

```
round(prop.table(table(wbcd$diagnosis)) * 100, digits = 1)
```

```
##
##    B    M
## 62.7 37.3
```

Vamos a cambiar la columna `diagnosis` por un factor.

```
wbcd$diagnosis = factor(wbcd$diagnosis, levels = c("B", "M"), labels = c("Benign", "Malignant"))
table(wbcd$diagnosis)
```

```
##
##    Benign Malignant
##       357       212
```

Ahora vamos a normalizar los valores numéricos de cada columna. Observemos en primer lugar el rango de algunas de las variables numéricas.

```
summary(wbcd[,c("radius_mean", "area_mean", "smoothness_mean")])
```

```
##   radius_mean       area_mean      smoothness_mean
##  Min.   : 6.981   Min.   : 143.5   Min.   :0.05263
##  1st Qu.:11.700   1st Qu.: 420.3   1st Qu.:0.08637
##  Median :13.370   Median : 551.1   Median :0.09587
##  Mean   :14.127   Mean   : 654.9   Mean   :0.09636
##  3rd Qu.:15.780   3rd Qu.: 782.7   3rd Qu.:0.10530
##  Max.   :28.110   Max.   :2501.0   Max.   :0.16340
```

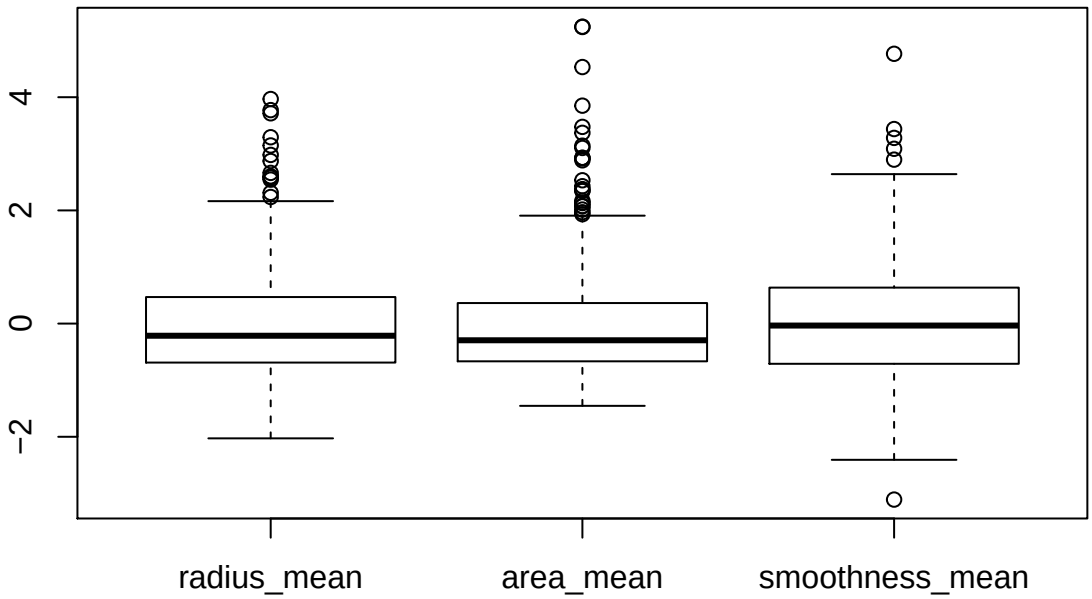
A continuación normalizamos los valores y nos aseguramos de que se han realizado los cambios satisfactoriamente.

```
wbcd_n = as.data.frame(lapply(wbcd[,2:31], scale, center = TRUE, scale = TRUE))

summary(wbcd_n[,c("radius_mean", "area_mean", "smoothness_mean")])
```

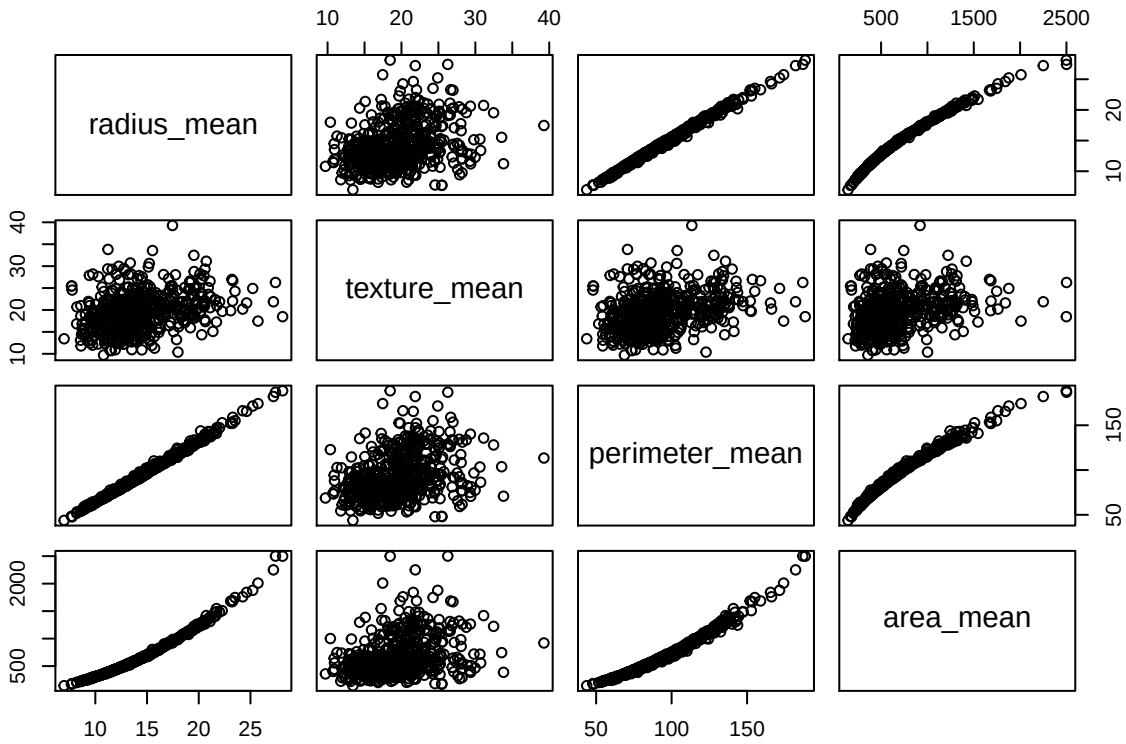
```
##   radius_mean        area_mean       smoothness_mean
##  Min.   :-2.0279   Min.   :-1.4532   Min.   :-3.10935
##  1st Qu.:-0.6888   1st Qu.:-0.6666   1st Qu.:-0.71034
##  Median :-0.2149   Median :-0.2949   Median :-0.03486
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000
##  3rd Qu.: 0.4690   3rd Qu.: 0.3632   3rd Qu.: 0.63564
##  Max.   : 3.9678   Max.   : 5.2459   Max.   : 4.76672
```

```
boxplot(wbcd_n[,c("radius_mean", "area_mean", "smoothness_mean")])
```

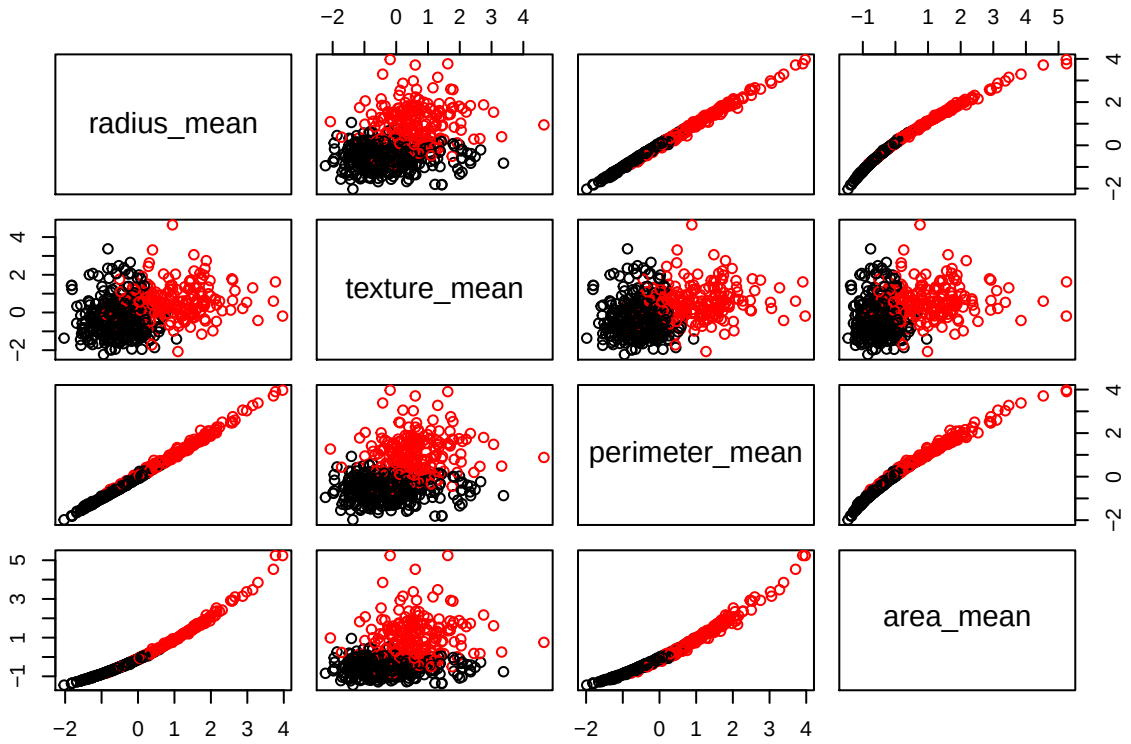


Si mostramos las gráficas de algunas de estas variables entre sí podremos confirmar que no existe diferencia en los datos (las gráficas son iguales). Lo mismo ocurre con las correlaciones entre las variables, la normalización de los datos no las modifica.

```
plot(wbcd[,2:5])
```




```
plot(wbcd_n[,1:4], col=wbcd[,1])
```




```
cor(wbcd[,2:5])
```

```
##                radius_mean texture_mean perimeter_mean area_mean
## radius_mean      1.0000000    0.3237819      0.9978553 0.9873572
## texture_mean     0.3237819    1.0000000      0.3295331 0.3210857
## perimeter_mean   0.9978553    0.3295331      1.0000000 0.9865068
## area_mean        0.9873572    0.3210857      0.9865068 1.0000000
```

```
cor(wbcd_n[,1:4])
```

```
##                radius_mean texture_mean perimeter_mean area_mean
## radius_mean      1.0000000    0.3237819      0.9978553 0.9873572
## texture_mean     0.3237819    1.0000000      0.3295331 0.3210857
## perimeter_mean   0.9978553    0.3295331      1.0000000 0.9865068
## area_mean        0.9873572    0.3210857      0.9865068 1.0000000
```

Creación de particiones

A continuación vamos a separar el dataset en conjuntos de train y de test, y a guardar sus etiquetas en una variable distinta.

```
shuffle_ds = sample(dim(wbcd_n)[1])
eightypct = (dim(wbcd_n)[1] * 80) %/% 100

wbcd_train = wbcd_n[shuffle_ds[1:eightypct], ]
wbcd_test = wbcd_n[shuffle_ds[(eightypct+1):dim(wbcd_n)[1]], ]

wbcd_train_labels = wbcd[shuffle_ds[1:eightypct], 1]
wbcd_test_labels = wbcd[shuffle_ds[(eightypct+1):dim(wbcd_n)[1]], 1]
```

Predecimos los valores de un modelo k-NN usando la función `knn` del paquete `class`.

```
library(class)
wbcd_test_pred = knn(train = wbcd_train, test = wbcd_test, cl = wbcd_train_labels, k=21)
table(wbcd_test_pred,wbcd_test_labels)
```

```
##               wbcd_test_labels
## wbcd_test_pred Benign Malignant
##      Benign        66        11
##      Malignant      0        37
```

Ejercicio 1. Probar diferentes configuraciones de k y hacer una comparación con los resultados.

Ahora vamos a probar el mismo funcionamiento usando las funciones `train` y `predict` del paquete `caret`. Esto nos facilitará la resolución del ejercicio planteado.

En primer lugar cargamos el paquete y probamos con algunas configuraciones de k, por ejemplo entre 1 y 10.

```
require(caret)
```

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
knnModel = train(wbcd_train, wbcd_train_labels, method="knn",
          metric="Accuracy", tuneGrid = data.frame(.k=1:10))
knnModel
```

```
## k-Nearest Neighbors
##
## 455 samples
##  30 predictor
##   2 classes: 'Benign', 'Malignant'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 455, 455, 455, 455, 455, 455, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.9501256  0.8901285
##    2  0.9492329  0.8882431
##    3  0.9489414  0.8873509
##    4  0.9472082  0.8831944
##    5  0.9547224  0.8997595
##    6  0.9526340  0.8949666
##    7  0.9563495  0.9033178
##    8  0.9548625  0.8999884
##    9  0.9558178  0.9021145
##   10  0.9563361  0.9033216
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 7.
```

Vemos que el mejor modelo, con el que se ha quedado la función train, es el 9-NN, sin embargo como esto puede provocar sobreajuste, vamos a escoger los 3 con mayor acierto para el estudio en test.

```
mejoresK = order(knnModel$results$Accuracy, decreasing = TRUE)[1:3]
mejoresK
```

```
## [1]  7 10  9
```

Obtenemos los modelos para cada uno de estos valores de k.

```
knnModel.1 = train(wbcd_train, wbcd_train_labels, method="knn",
                metric="Accuracy", tuneGrid = data.frame(.k=mejoresK[1]))

knnModel.2 = train(wbcd_train, wbcd_train_labels, method="knn",
                metric="Accuracy", tuneGrid = data.frame(.k=mejoresK[2]))

knnModel.3 = train(wbcd_train, wbcd_train_labels, method="knn",
                metric="Accuracy", tuneGrid = data.frame(.k=mejoresK[3]))
```

A continuación predecimos las etiquetas para la variable de salida con los datos de test.

```
knnPred.1 = predict(knnModel.1, wbcd_test)
knnPred.2 = predict(knnModel.2, wbcd_test)
knnPred.3 = predict(knnModel.3, wbcd_test)
```

Y una vez obtenidas las etiquetas, comprobamos el acierto logrado por cada uno de los modelos en el conjunto de test.

```
postResample(knnPred.1, wbcd_test_labels)
```

```
##  Accuracy     Kappa
## 0.9473684 0.8901734
```

```
postResample(knnPred.2, wbcd_test_labels)
```

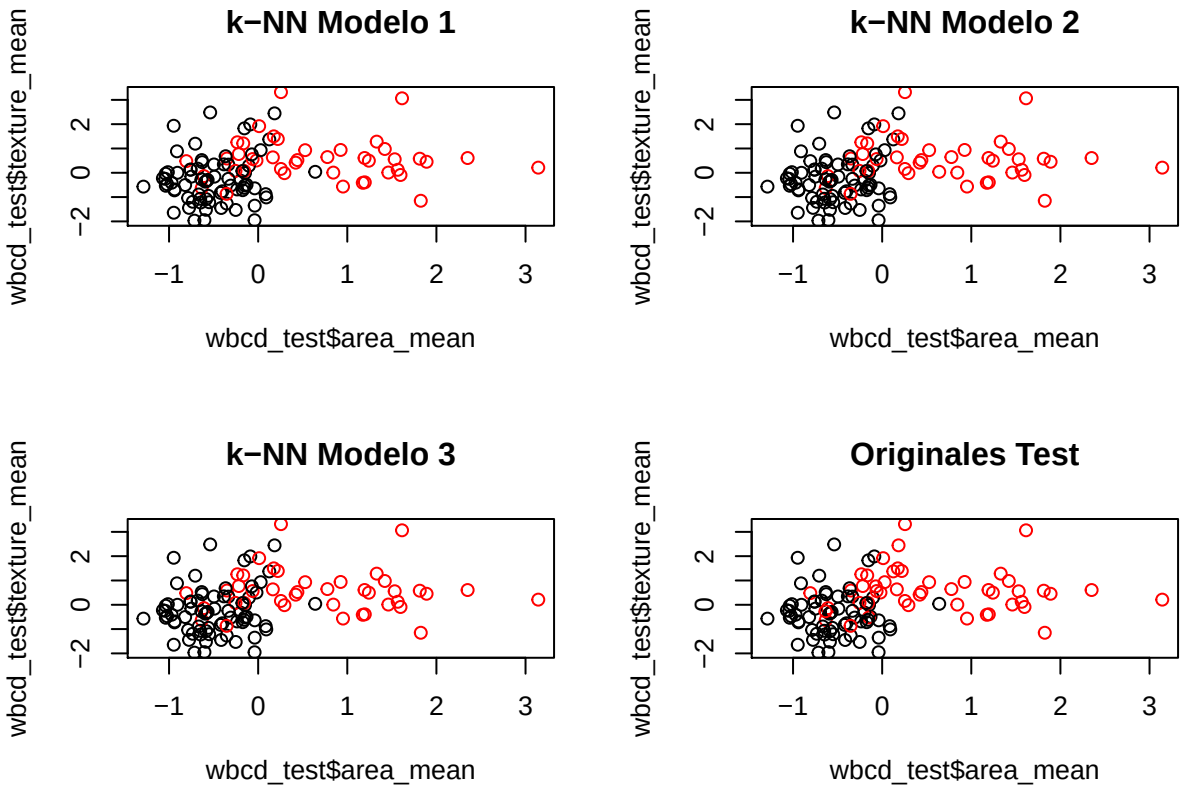
```
##  Accuracy     Kappa
## 0.9210526 0.8347826
```

```
postResample(knnPred.3, wbcd_test_labels)
```

```
##  Accuracy     Kappa
## 0.9385965 0.8714976
```

También podemos comparar visualmente los resultados de los modelos entre sí en una gráfica. Además se van a añadir las etiquetas originales en una última gráfica que sirva como referencia.

```
par(mfrow=c(2,2))
plot(wbcd_test$texture_mean~wbcd_test$area_mean,col=knnPred.1, main="k-NN Modelo 1")
plot(wbcd_test$texture_mean~wbcd_test$area_mean,col=knnPred.2, main="k-NN Modelo 2")
plot(wbcd_test$texture_mean~wbcd_test$area_mean,col=knnPred.3, main="k-NN Modelo 3")
plot(wbcd_test$texture_mean~wbcd_test$area_mean,col=wbcd_test_labels, main="Originales Test")
```




```
par(mfrow=c(1,1))
```

Ejercicio 2. Usando el dataset Smarket, realizar una 10-fold cross-validation con regresión logística.

```
library(ISLR)
head(Smarket)
```

```
##   Year   Lag1   Lag2   Lag3   Lag4   Lag5 Volume  Today Direction
## 1 2001  0.381 -0.192 -2.624 -1.055  5.010 1.1913  0.959        Up
## 2 2001  0.959  0.381 -0.192 -2.624 -1.055 1.2965  1.032        Up
## 3 2001  1.032  0.959  0.381 -0.192 -2.624 1.4112 -0.623      Down
## 4 2001 -0.623  1.032  0.959  0.381 -0.192 1.2760  0.614        Up
## 5 2001  0.614 -0.623  1.032  0.959  0.381 1.2057  0.213        Up
## 6 2001  0.213  0.614 -0.623  1.032  0.959 1.3491  1.392        Up
```

Una vez cargado el dataset, entrenamos un modelo `glm` (regresión logística) usando de nuevo la función `train` del paquete caret. Entrenaremos con todas las columnas menos la columna `Direction`, que contiene las etiquetas, y la columna `Today`, de la que se extrae el valor de Direction y que sería perjudicial para nuestro modelo por provocar sobreaprendizaje.

```
glmFit = train(Smarket[,-8:-9], y=Smarket[,9], method="glm",
         preProcess=c("center", "scale"), tuneLength=10,
         control=glm.control(maxit=500), trControl=trainControl(method = "cv"))
glmFit
```

```
## Generalized Linear Model
##
## 1250 samples
```

```
##    7 predictor
##    2 classes: 'Down', 'Up'
##
## Pre-processing: centered (7), scaled (7)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1125, 1125, 1124, 1125, 1125, 1125, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.5288326  0.04559167
```

Vemos que el modelo obtenido no es muy bueno, apenas superando el 50% de acierto, por lo que no lo consideraremos adecuado para hacer una predicción sobre estos datos.

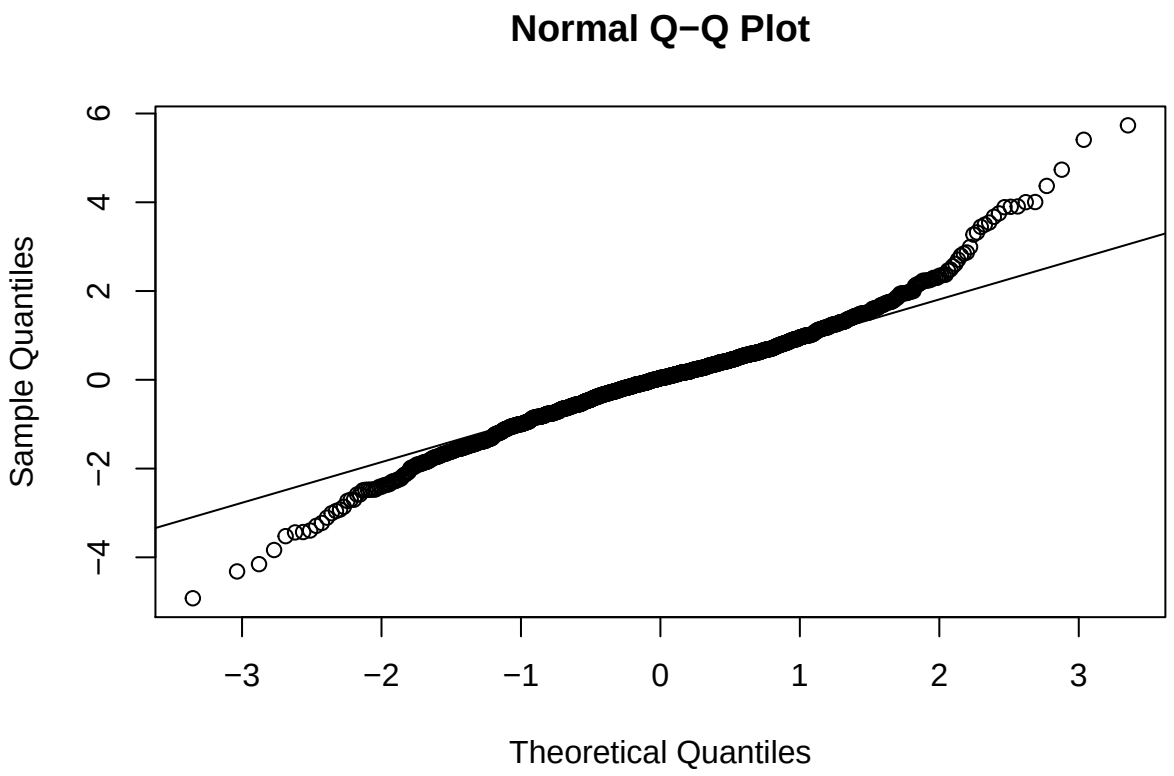
Ejercicio 3. Probar LDA con todas las variables Lag del dataset Smarket. Hacer una rápida comparativa entre regresión logística y LDA. Probar QDA y comparar los tres métodos con gráficas.

Antes de utilizar LDA vamos a comprobar que las variables que serán objeto de nuestro estudio siguen una distribución normal. Para ello vamos a utilizar el test de Saphiro-Wilk.

```
shapiro.test(Smarket$Lag1)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Smarket$Lag1
## W = 0.97219, p-value = 8.889e-15
```

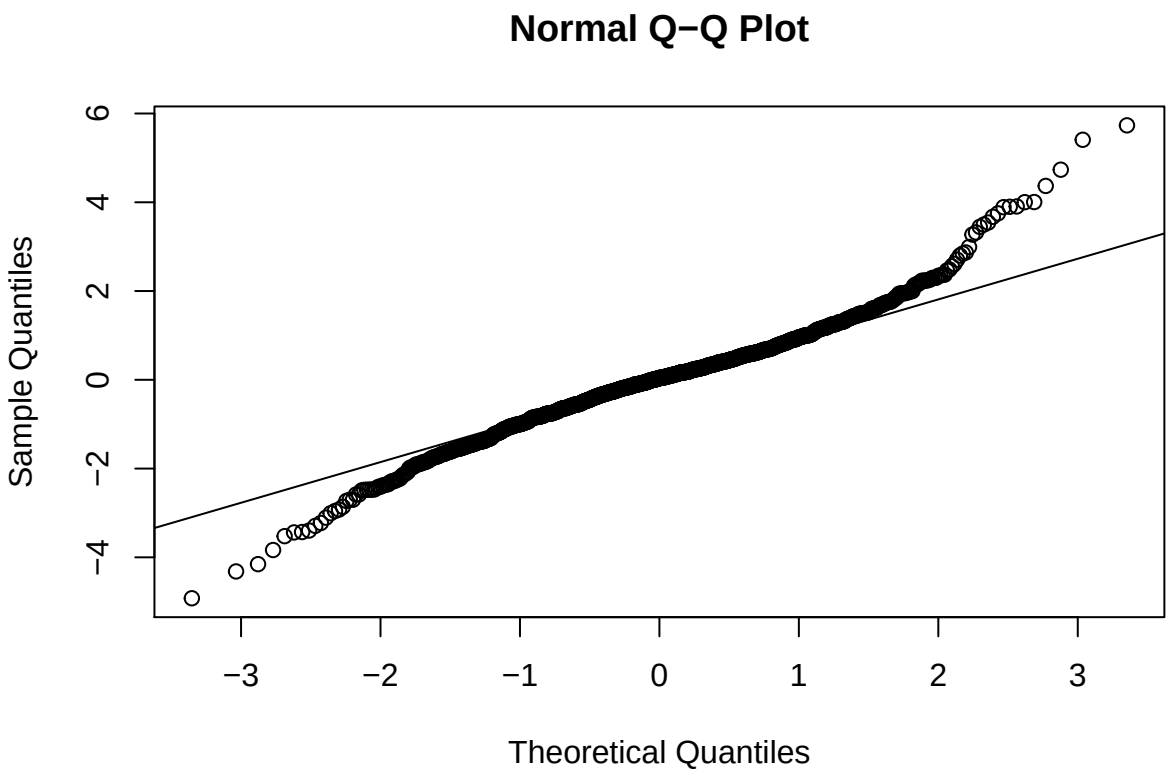
```
qqnorm(y=Smarket$Lag1)
qqline(y=Smarket$Lag1)
```




```
shapiro.test(Smarket$Lag2)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Smarket$Lag2
## W = 0.97217, p-value = 8.798e-15
```

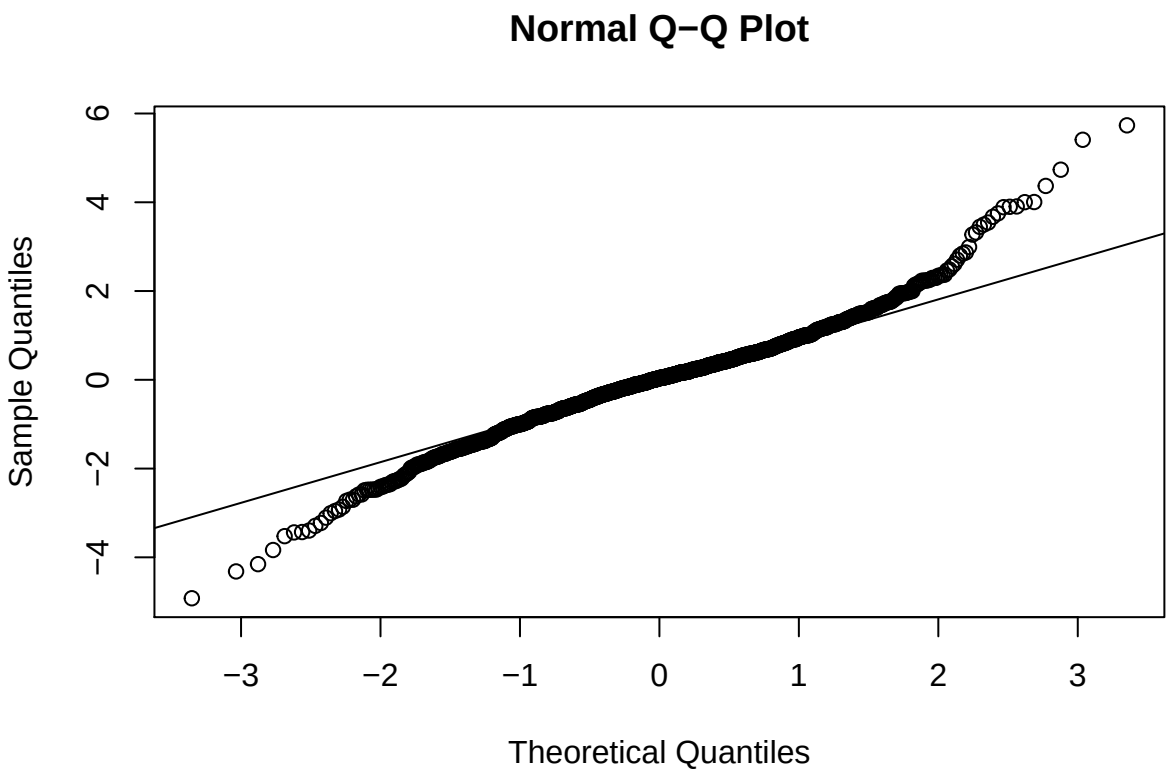
```
qqnorm(y=Smarket$Lag2)
qqline(y=Smarket$Lag2)
```




```
shapiro.test(Smarket$Lag3)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Smarket$Lag3
## W = 0.9724, p-value = 1.035e-14
```

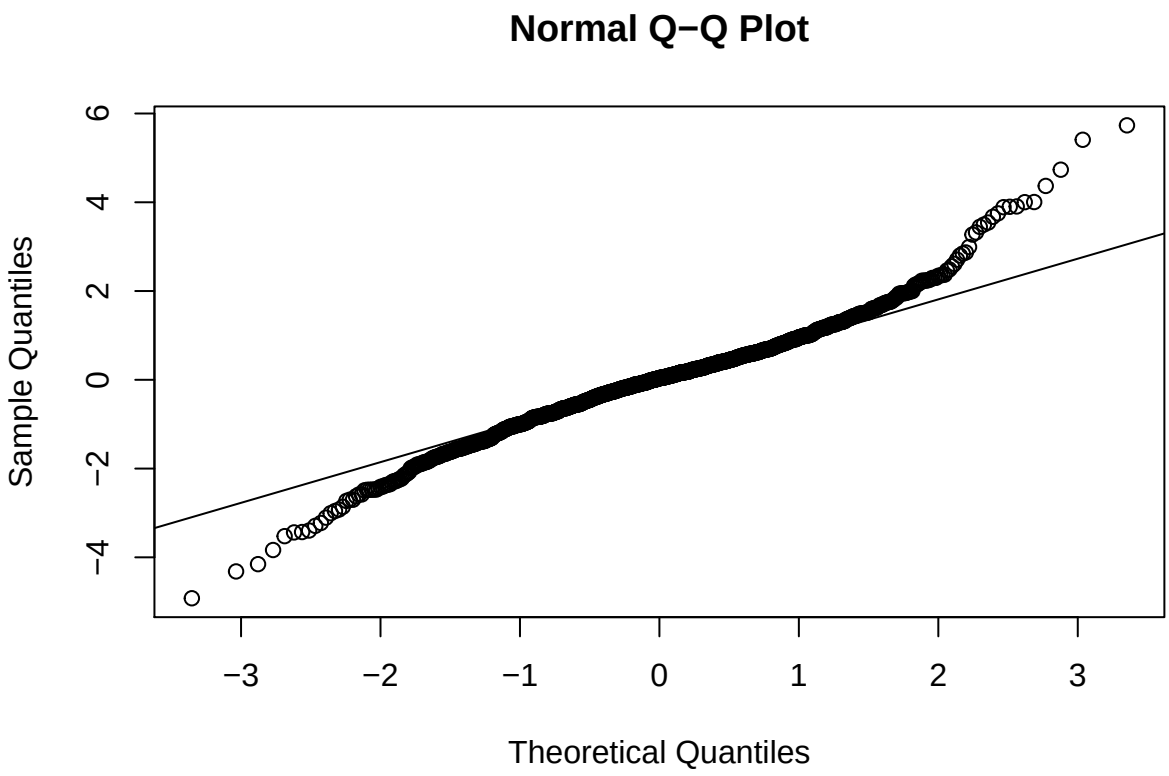
```
qqnorm(y=Smarket$Lag3)
qqline(y=Smarket$Lag3)
```




```
shapiro.test(Smarket$Lag4)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Smarket$Lag4
## W = 0.97242, p-value = 1.049e-14
```

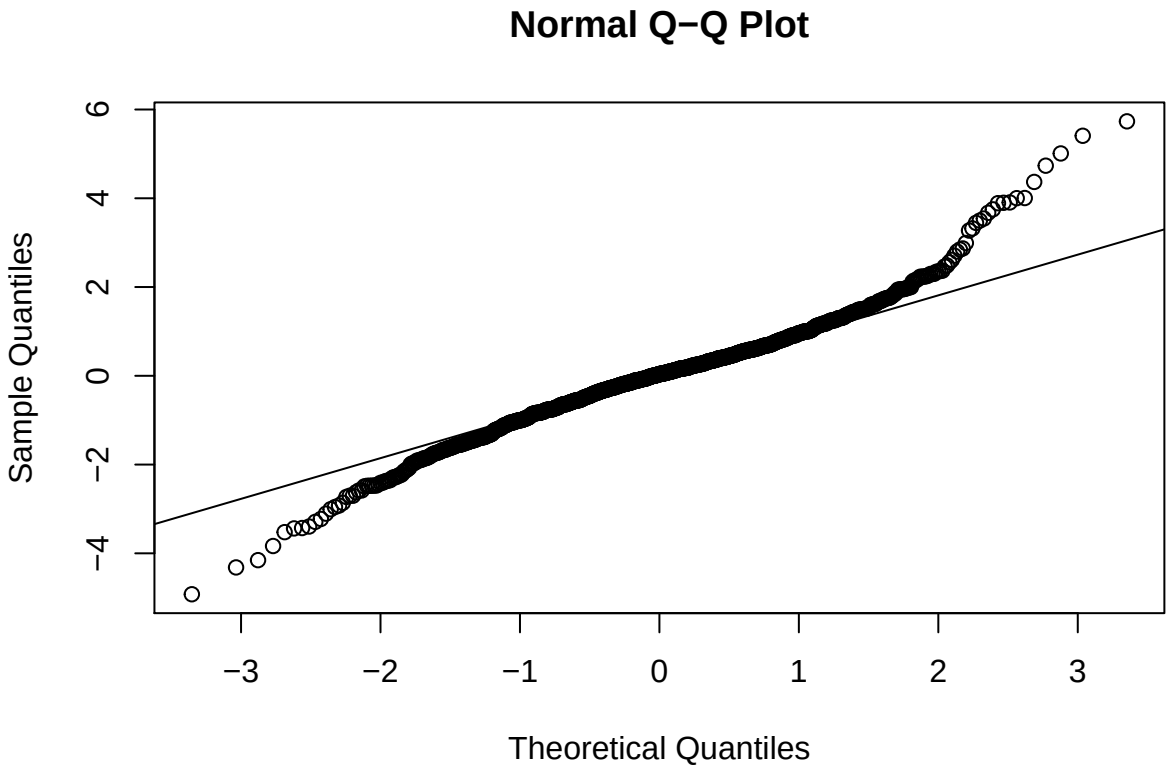
```
qqnorm(y=Smarket$Lag4)
qqline(y=Smarket$Lag4)
```




```
shapiro.test(Smarket$Lag5)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Smarket$Lag5
## W = 0.97011, p-value = 2.149e-15
```

```
qqnorm(y=Smarket$Lag5)
qqline(y=Smarket$Lag5)
```



De acuerdo a los valores cercanos a 1 obtenidos como resultado del test de Saphiro-Wilk, y en relación a lo observado en las gráficas, podemos afirmar que las 5 variables Lag están distribuidas normalmente. El siguiente paso es asegurarnos de que la varianza es similar entre las distintas variables.

```
var(Smarket$Lag1)
```

```
## [1] 1.291175
```

```
var(Smarket$Lag2)
```

```
## [1] 1.291133
```

```
var(Smarket$Lag3)
```

```
## [1] 1.296644
```

```
var(Smarket$Lag4)
```

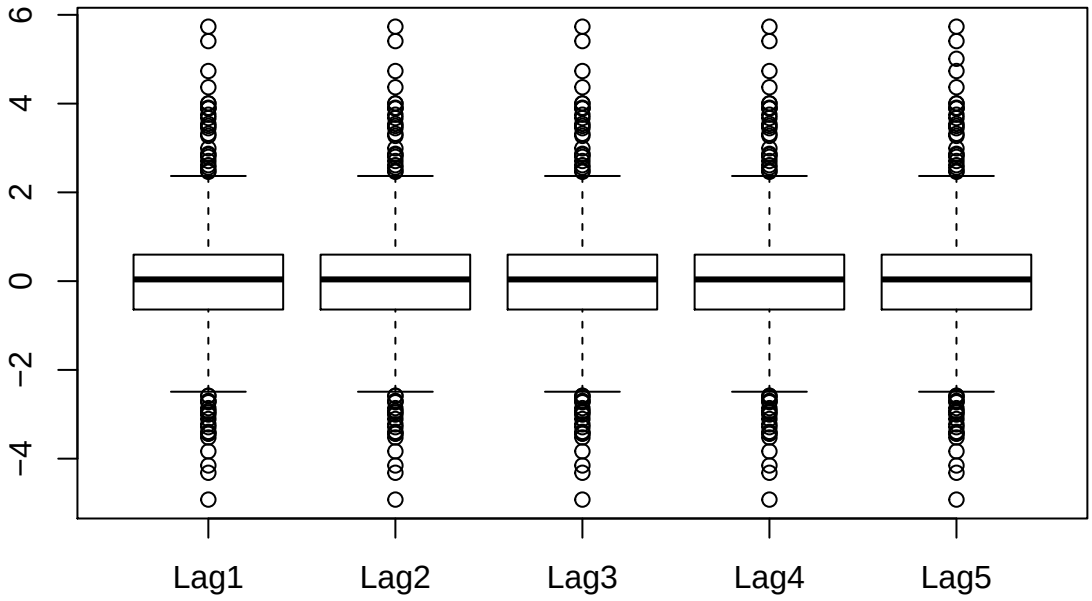
```
## [1] 1.296806
```

```
var(Smarket$Lag5)
```

```
## [1] 1.316871
```

También podemos confirmarlo mediante una gráfica de tipo “box-plot”.

```
boxplot(Smarket[2:6])
```



Las gráficas son prácticamente equivalentes por lo que aseguramos la similitud entre las variables. Ahora podemos realizar LDA. Lo haremos con los años previos a 2005 para luego comprobar la predicción sobre 2005.

```
require(MASS)
```

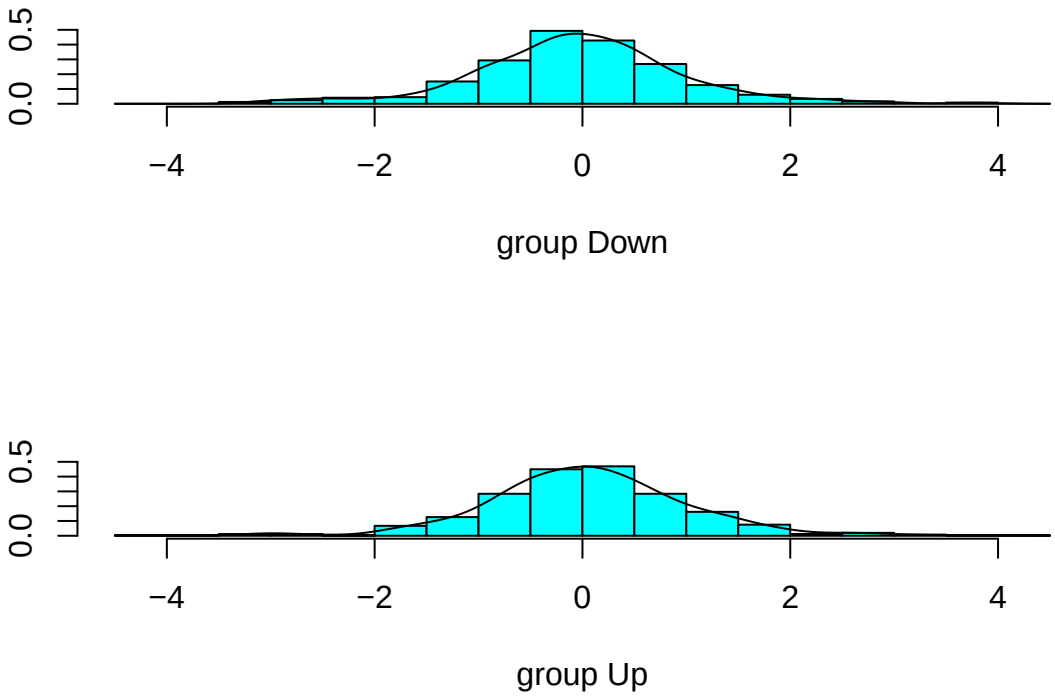
```
## Loading required package: MASS
```

```
lda.fit <- lda(Direction~Lag1+Lag2+Lag3+Lag4+Lag5, data=Smarket, subset=Year<2005)
lda.fit
```

```
## Call:
## lda(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5, data = Smarket,
##     subset = Year < 2005)
##
## Prior probabilities of groups:
##     Down       Up
## 0.491984 0.508016
##
## Group means:
##             Lag1        Lag2         Lag3         Lag4          Lag5
## Down  0.04279022  0.03389409 -0.009806517 -0.010598778  0.0043665988
## Up   -0.03954635 -0.03132544  0.005834320  0.003110454 -0.0006508876
##
## Coefficients of linear discriminants:
##              LD1
## Lag1 -0.63046918
## Lag2 -0.50221745
## Lag3  0.10142974
## Lag4  0.09725317
## Lag5 -0.03685767
```

Visualicemos los datos del modelo para ambas clases.

```
plot(lda.fit, type="both")
```



Vamos a realizar la predicción del año 2005 con el modelo anterior.

```
Smarket.2005 = subset(Smarket,Year==2005)
lda.pred = predict(lda.fit, Smarket.2005)
```

Obtenemos la matriz de confusión y el porcentaje de acierto de este modelo para el año 2005.

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

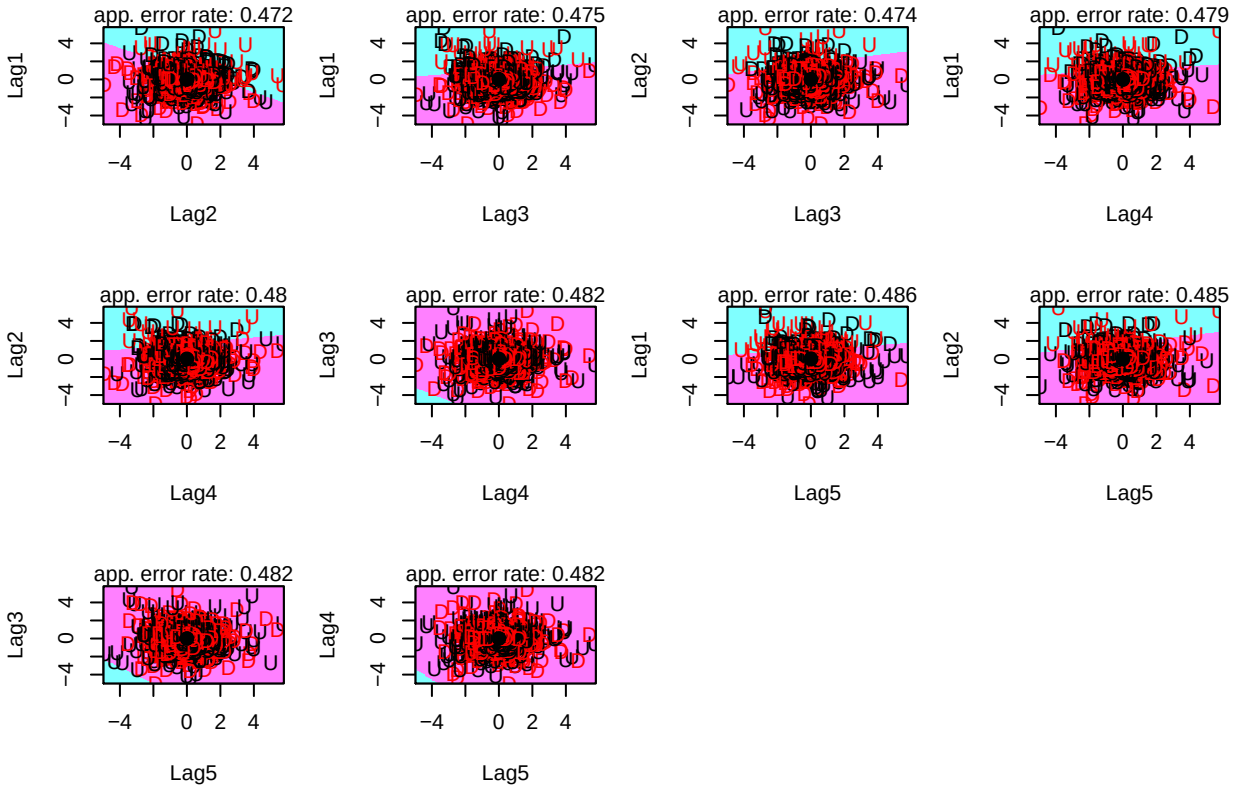
```
##
##        Down  Up
##   Down   37  30
##   Up     74 111
```

```
ldaPred = mean(lda.pred$class==Smarket.2005$Direction)
ldaPred
```

```
## [1] 0.5873016
```

Como podemos observar no existe un acierto muy elevado, apenas de un 58%. Con la librería `klaR` podemos mostrar las variables una frente a otra y enseñar cómo divide el espacio el modelo.

```
library(klaR)
partimat(Direction~Lag1+Lag2+Lag3+Lag4+Lag5, data=Smarket, method="lda")
```



Vamos a comparar estos resultados con los obtenidos con regresión logística.

```
glmFit$results$Accuracy
```

```
## [1] 0.5288326
```

```
ldaPred
```

```
## [1] 0.5873016
```

En base a estos resultados podemos afirmar que LDA se ha comportado mejor que regresión logística para este conjunto de datos.

Vamos ahora a probar los resultados con QDA. Para ello comprobamos antes si los datos dentro de la propia clase también se distribuyen de forma normal.

```
Smarket.up = Smarket[Smarket$Direction == "Up",]
Smarket.down = Smarket[Smarket$Direction == "Down",]

var(Smarket.up$Lag1)
```

```
## [1] 1.279137
```

```
var(Smarket.up$Lag2)
```

```
## [1] 1.247171
```

```
var(Smarket.up$Lag3)
```

```
## [1] 1.277851
```

```r
var(Smarket.up$Lag4)
```

```
## [1] 1.220924
```

```r
var(Smarket.up$Lag5)
```

```
## [1] 1.363483
```

```r
var(Smarket.down$Lag1)
```

```
## [1] 1.302041
```

```r
var(Smarket.down$Lag2)
```

```
## [1] 1.339052
```

```r
var(Smarket.down$Lag3)
```

```
## [1] 1.318933
```

```r
var(Smarket.down$Lag4)
```

```
## [1] 1.380605
```

```r
var(Smarket.down$Lag5)
```

```
## [1] 1.268803
```

A continuación procedemos a utilizar QDA.

```r
qda.fit = qda(Direction~Lag1+Lag2+Lag3+Lag4+Lag5, data=Smarket, subset=Year<2005)
qda.fit
```

```
## Call:
## qda(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5, data = Smarket,
##     subset = Year < 2005)
##
## Prior probabilities of groups:
##     Down       Up
## 0.491984 0.508016
##
## Group means:
##             Lag1        Lag2         Lag3         Lag4          Lag5
## Down  0.04279022  0.03389409 -0.009806517 -0.010598778  0.0043665988
## Up   -0.03954635 -0.03132544  0.005834320  0.003110454 -0.0006508876
```

Con el modelo generado podemos predecir los datos para el año 2005.

```r
qda.pred = predict(qda.fit, Smarket.2005)

table(qda.pred$class, Smarket.2005$Direction)
```

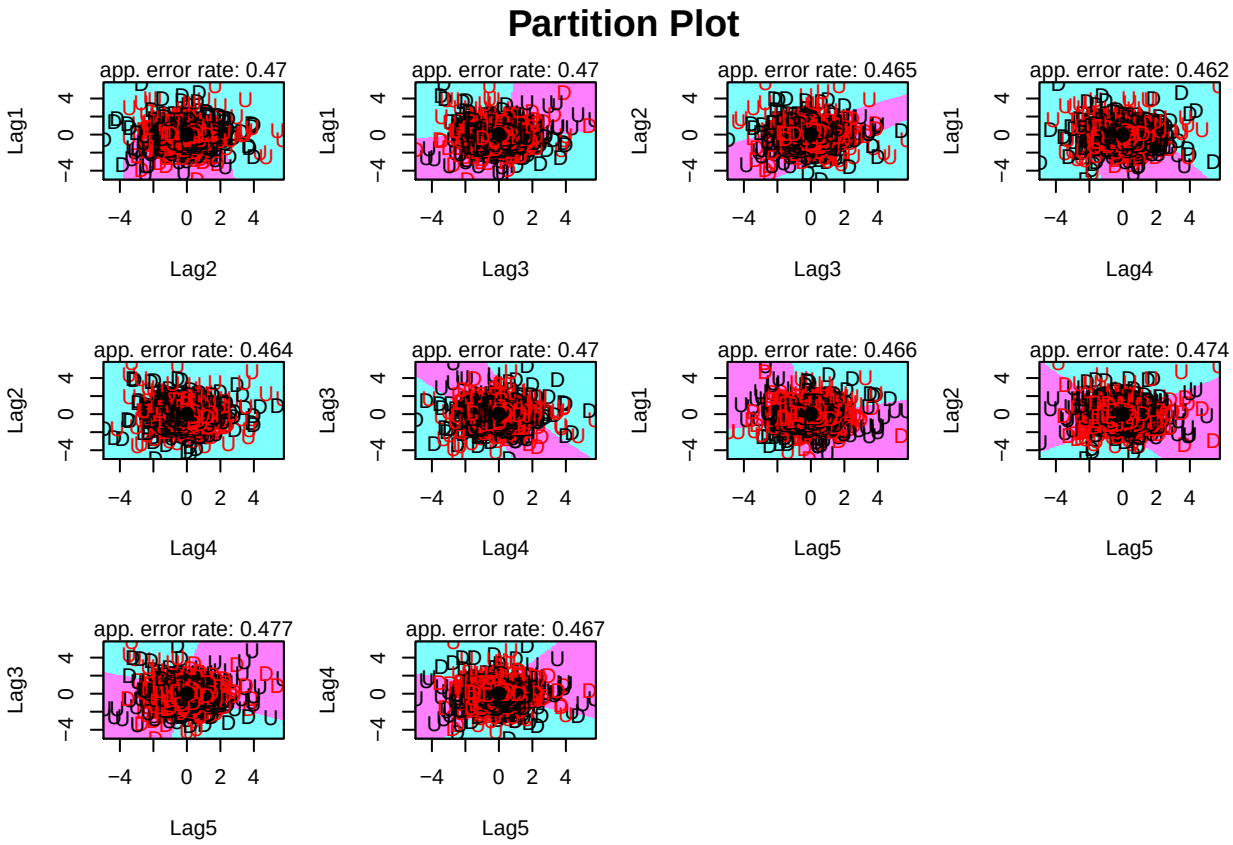
```
##
##        Down  Up
##   Down   37  35
##   Up     74 106
```

```r
qdaPred = mean(qda.pred$class==Smarket.2005$Direction)
qdaPred
```

```
## [1] 0.5674603
```

QDA no aporta mejores resultados que LDA, por tanto, a pesar de que este no es un buen modelo, sí es el mejor de los tres que se han probado en este dataset. Podemos echar un vistazo a las gráficas generadas con el paquete `klaR` para el modelo QDA.

```
partimat(Direction~Lag1+Lag2+Lag3+Lag4+Lag5, data=Smarket, method="qda")
```



Ejercicio 4. Usando la información del dataset `clasif_train_alumnos.csv`, comparar LDA y QDA con Wilcoxon, realizar una comparativa múltiple usando Friedman y usar Holm para ver si hay un algoritmo ganador.

En primer lugar cargamos el nuevo dataset.

```
alumnos = read.csv("Datos/clasif_train_alumnos.csv")
tabla.train = cbind(alumnos[,2:dim(alumnos)[2]])
colnames(tabla.train) = names(alumnos)[2:dim(alumnos)[2]]
rownames(tabla.train) = alumnos[,1]
```

Con los datos guardados, aplicamos el test de Wilcoxon entre LDA y QDA.

```
LDAvsQDA = wilcox.test(tabla.train$out_train_lda, tabla.train$out_train_qda,
           alternative="two.sided", paired=TRUE)
Rmas = LDAvsQDA$statistic
pvalue = LDAvsQDA$p.value
LDAvsQDA = wilcox.test(tabla.train$out_train_qda, tabla.train$out_train_lda,
              alternative="two.sided", paired=TRUE)
Rmenos = LDAvsQDA$statistic
Rmas
```

```
##  V
## 68
```

```
Rmenos
```

```
##   V
## 142
```

```
pvalue
```

```
## [1] 0.1768532
```

Dados estos resultados podemos afirmar con un 82% de confianza que existen diferencias suficientemente significativas entre LDA y QDA para este dataset.

A continuación realizaremos el test de Friedman con los datos anteriores.

```
friedman.test(as.matrix(tabla.train))
```

```
##
##  Friedman rank sum test
##
## data:  as.matrix(tabla.train)
## Friedman chi-squared = 1.3, df = 2, p-value = 0.522
```

Según el test de Friedman sin embargo no podemos asegurar lo anterior con una confianza elevada, estos algoritmos tan sólo serían distintos de forma significativa al 48% de confianza.

Por último, comparemos los algoritmos mediante el test de Holm.

```
tam = dim(tabla.train)
groups = rep(1:tam[2], each=tam[1])
pairwise.wilcox.test(as.matrix(tabla.train), groups, p.adjust = "holm", paired = TRUE)
```

```
##
##  Pairwise comparisons using Wilcoxon signed rank test
##
## data:  as.matrix(tabla.train) and groups
##
##   1    2
## 2 0.65 -
## 3 0.59 0.53
##
## P value adjustment method: holm
```

Dado que todos los valores de la tabla resultante son cercanos al 50% no podemos afirmar que exista una gran diferencia entre los algoritmos. Por tanto, no existe un ganador claro entre ellos.