Ejercicios Clasificaci?n

December 6, 2018

0.1 Ejercios Clasificación

Alberto Armijo Ruiz

0.1.1 Ejercicio 1

Probar diferentes valores para k y compararlos. Hacer un plot con los resultados.

In [2]:
```
# Primero leemos el archivo que contiene el dataset de breast cancer
wbcd = read.csv("wisc_bc_data.csv",stringsAsFactors = FALSE)
str(wbcd)
head(wbcd)
```

```
'data.frame':        569 obs. of  32 variables:
 $ id                : int  87139402 8910251 905520 868871 9012568 906539 925291 87880 862989 89
 $ 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 ...
```

id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
87139402	B	12.32	12.39	78.85	464.1	0.10280
8910251	B	10.60	18.95	69.28	346.4	0.09688
905520	B	11.04	16.83	70.92	373.2	0.10770
868871	B	11.28	13.39	73.00	384.8	0.11640
9012568	B	15.19	13.21	97.65	711.8	0.07963
906539	B	11.57	19.04	74.20	409.7	0.08546

In [3]:
```
# Quitamos el id ya que no nos sirve de nada para predecir.
wbcd = wbcd[,-1]

# Transformamos los datos de la variable diagnosis a un factor.
wbcd$diagnosis = factor(wbcd$diagnosis, levels=c("B","M"),
                        labels = c("Benign","Malignant"))
# Miramos el número de datos de cada tipo.
table(wbcd$diagnosis)
```

```
   Benign Malignant
      357       212
```

In [4]:
```
# Normalizamos los datos para poder aplicar knn.
wbcd_n <- as.data.frame(lapply(wbcd[,2:31],
                               scale, center = TRUE, scale = TRUE))
```

In [5]:
```
# Creamos datasets de train y test
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]], ]

# Creamos también labels para ambos subconjuntos.
wbcd_train_labels <- wbcd[shuffle_ds[1:eightypct], 1]
wbcd_test_labels <- wbcd[shuffle_ds[(eightypct+1):dim(wbcd_n)[1]], 1]
```

In [6]:
```
library(class)
wbcd_test_pred <- knn(train = wbcd_train, test = wbcd_test, cl = wbcd_train_labels, k=
wbcd_test_pred
```

1. Malignant 2. Benign 3. Malignant 4. Benign 5. Benign 6. Benign 7. Benign 8. Malignant 9. Malignant 10. Benign 11. Malignant 12. Benign 13. Benign 14. Malignant 15. Benign 16. Benign 17. Malignant 18. Malignant 19. Malignant 20. Malignant 21. Benign 22. Benign 23. Benign 24. Malignant 25. Malignant 26. Malignant 27. Malignant 28. Benign 29. Benign 30. Benign 31. Benign 32. Malignant 33. Benign 34. Benign 35. Benign 36. Benign 37. Benign 38. Benign 39. Malignant 40. Malignant 41. Benign 42. Malignant 43. Benign 44. Benign 45. Benign 46. Malignant 47. Malignant 48. Benign 49. Benign 50. Malignant 51. Malignant 52. Malignant 53. Benign 54. Benign 55. Malignant 56. Malignant 57. Benign 58. Benign 59. Malignant 60. Benign 61. Benign 62. Benign 63. Benign 64. Malignant 65. Malignant 66. Benign 67. Benign 68. Malignant 69. Malignant 70. Benign 71. Malignant 72. Malignant 73. Benign 74. Benign 75. Benign 76. Benign 77. Benign 78. Malignant 79. Malignant 80. Benign 81. Malignant 82. Malignant 83. Malignant 84. Benign 85. Benign 86. Malignant 87. Benign 88. Malignant 89. Benign 90. Benign 91. Benign 92. Benign 93. Malignant 94. Benign 95. Malignant 96. Benign 97. Benign 98. Benign 99. Benign 100. Malignant 101. Benign 102. Benign 103. Benign 104. Benign 105. Malignant 106. Malignant 107. Benign 108. Benign 109. Malignant 110. Benign 111. Benign 112. Malignant 113. Benign 114. Malignant

Levels: 1. 'Benign' 2. 'Malignant'

```
In [7]: # Evaluating model performance
        table(wbcd_test_pred,wbcd_test_labels)
```

```
              wbcd_test_labels
wbcd_test_pred Benign Malignant
     Benign        64         3
     Malignant      1        46
```

```
In [8]: require(caret)
        knnModel <- train(x = wbcd_train, y = wbcd_train_labels, method = "knn")
        class(knnModel)
        knnModel
```

```
Loading required package: caret
Loading required package: lattice
Loading required package: ggplot2
```

'train'

```
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
  5  0.9521769  0.8951747
  7  0.9587139  0.9094152
  9  0.9563613  0.9042244

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 7.
```

```
In [9]: knnModel <- train(wbcd_train, wbcd_train_labels, method="knn", metric="Accuracy", tune
        knnModel
        knnModel$results
```

```
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  0.9478945  0.8847191
   2  0.9463138  0.8812730
   3  0.9488789  0.8871577
   4  0.9502554  0.8900188
   5  0.9535576  0.8969792
   6  0.9570826  0.9046823
   7  0.9574269  0.9051750
   8  0.9546111  0.8986517
   9  0.9563040  0.9024635
  10  0.9550870  0.8998954
  11  0.9557871  0.9015039
  12  0.9562668  0.9023992
  13  0.9566278  0.9027401
  14  0.9557234  0.9008844
  15  0.9545654  0.8983665

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 7.
```

k	Accuracy	Kappa	AccuracySD	KappaSD
1	0.9478945	0.8847191	0.01275406	0.03034491
2	0.9463138	0.8812730	0.01458878	0.03233081
3	0.9488789	0.8871577	0.01617309	0.03435837
4	0.9502554	0.8900188	0.01726663	0.03745958
5	0.9535576	0.8969792	0.01453045	0.03136439
6	0.9570826	0.9046823	0.01526867	0.03360832
7	0.9574269	0.9051750	0.01347818	0.02965065
8	0.9546111	0.8986517	0.01476031	0.03378720
9	0.9563040	0.9024635	0.01260454	0.02787252
10	0.9550870	0.8998954	0.01298011	0.02822099
11	0.9557871	0.9015039	0.01359157	0.02961247
12	0.9562668	0.9023992	0.01347443	0.02935660
13	0.9566278	0.9027401	0.01222972	0.02813461
14	0.9557234	0.9008844	0.01200884	0.02611681
15	0.9545654	0.8983665	0.01153953	0.02477225

In [10]:
```
require(caret)
knnModel <- train(x = wbcd[shuffle_ds[1:eightypct],-1], y = wbcd[shuffle_ds[1:eightypc
class(knnModel)
knnModel
```

'train'

```
k-Nearest Neighbors

455 samples
 30 predictor
  2 classes: 'Benign', 'Malignant'

Pre-processing: centered (30), scaled (30)
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 455, 455, 455, 455, 455, 455, ...
Resampling results across tuning parameters:

  k  Accuracy   Kappa
  5  0.9548653  0.9003593
  7  0.9563098  0.9035668
  9  0.9589189  0.9090480

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 9.
```

In [11]:
```
knnPred <- predict(knnModel, newdata = wbcd[shuffle_ds[(eightypct+1):dim(wbcd_n)[1]],
knnPred
```

1. Malignant 2. Benign 3. Malignant 4. Benign 5. Benign 6. Benign 7. Benign 8. Malignant 9. Malignant 10. Benign 11. Malignant 12. Benign 13. Benign 14. Malignant 15. Benign 16. Benign 17. Malignant 18. Malignant 19. Malignant 20. Malignant 21. Benign 22. Benign 23. Benign 24. Malignant

25. Malignant 26. Malignant 27. Malignant 28. Benign 29. Benign 30. Benign 31. Benign 32. Malignant 33. Benign 34. Benign 35. Benign 36. Benign 37. Benign 38. Benign 39. Malignant 40. Malignant 41. Benign 42. Malignant 43. Benign 44. Benign 45. Benign 46. Malignant 47. Malignant 48. Benign 49. Benign 50. Malignant 51. Malignant 52. Malignant 53. Benign 54. Benign 55. Malignant 56. Malignant 57. Benign 58. Benign 59. Malignant 60. Benign 61. Benign 62. Benign 63. Benign 64. Malignant 65. Malignant 66. Benign 67. Benign 68. Malignant 69. Malignant 70. Benign 71. Malignant 72. Malignant 73. Benign 74. Benign 75. Benign 76. Benign 77. Benign 78. Malignant 79. Malignant 80. Benign 81. Malignant 82. Malignant 83. Malignant 84. Benign 85. Malignant 86. Malignant 87. Benign 88. Malignant 89. Benign 90. Benign 91. Benign 92. Benign 93. Malignant 94. Benign 95. Malignant 96. Benign 97. Benign 98. Benign 99. Benign 100. Malignant 101. Benign 102. Benign 103. Benign 104. Benign 105. Malignant 106. Malignant 107. Benign 108. Benign 109. Malignant 110. Benign 111. Benign 112. Malignant 113. Benign 114. Malignant

Levels: 1. 'Benign' 2. 'Malignant'

In [12]:
```
postResample(pred = knnPred, obs = wbcd[shuffle_ds[(eightypct+1):dim(wbcd_n)[1]], 1])
```

Accuracy 0.973684210526316 **Kappa** 0.946175637393768

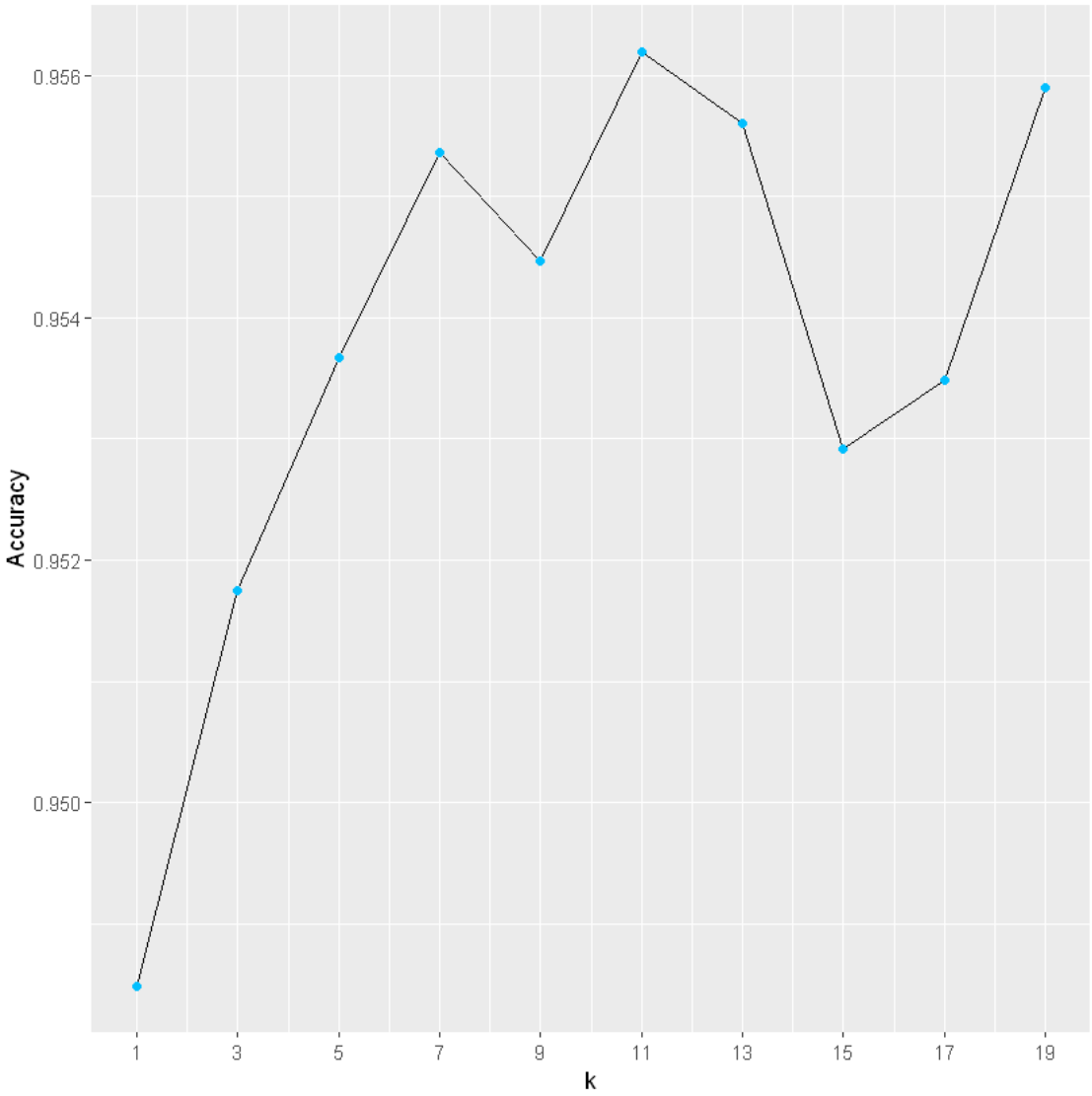
In [13]:
```
# Ejercicio, probar con diferentes valores de k y
# dibujar un plot con los resultados.

# Primero crearemos una lista con los nivesles de k que nos interesan
# para ello iremos desde 1 hasta 20, quedandonos solamente con los valores
# impares, ya que con valores impares podría darse empates con knn.

ks = 1:20
ks = ks[ks%%2 != 0]

# Ahora crearemos un modelo que utilice esta lista de puntos y nos devuelva
# los valores de accuracy para cada k
knnModel <- train(wbcd_train, wbcd_train_labels, method="knn", metric="Accuracy", tune
knnModel

# Obtenemos los resultados del modelo.
resultados = knnModel$results[,c("k","Accuracy")]
resultados
```

```
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  0.9484865  0.8870499
  3  0.9517521  0.8937340
  5  0.9536736  0.8974576
  7  0.9553670  0.9007900
  9  0.9544674  0.8986978
 11  0.9561944  0.9024569
 13  0.9556020  0.9008999
 15  0.9529198  0.8944641
 17  0.9534857  0.8958782
 19  0.9559035  0.9010304

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 11.
```

k	Accuracy
1	0.9484865
3	0.9517521
5	0.9536736
7	0.9553670
9	0.9544674
11	0.9561944
13	0.9556020
15	0.9529198
17	0.9534857
19	0.9559035

In [14]:
```
library(ggplot2)
ggplot(resultados,aes(x=k,y=Accuracy))+geom_line()+
geom_point(col="deepskyblue")+
scale_x_continuous(breaks=resultados$k)
```



Según lo que se puede ver en la gráfica, el mejor valor para k es 5 para este conjunto de datos, ya que obtiene mejores resultados que todos los otros k. Otros valores cercanos como 3,7 o 9 obtienen también resultados buenos (Además son valores usuales para knn). Para valores grandes, se puede ver que los resultados son peores, por ello no se suelen elegir normalmente. (Con otras ejecuciones los resultados de la gráfica pueden ser diferentes)

0.1.2 Ejercicio 2

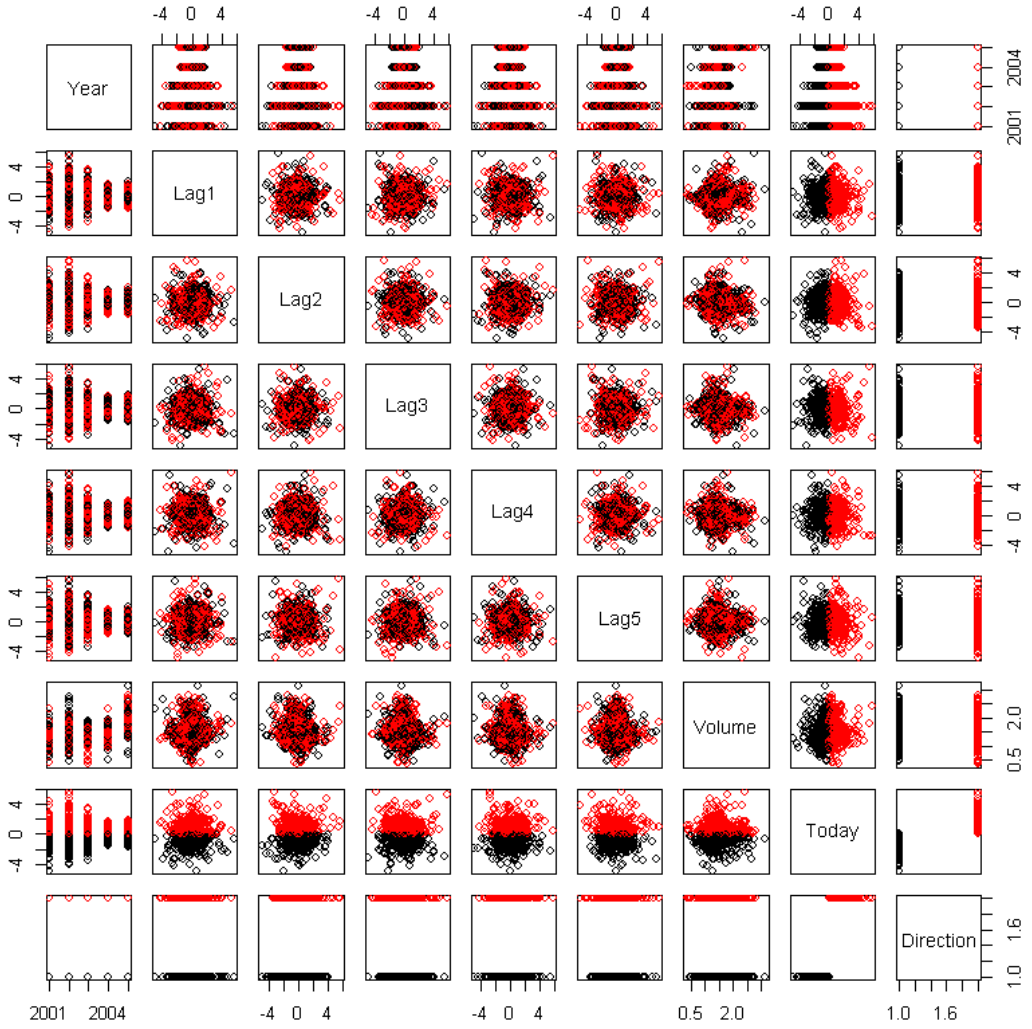
Hacer un 10 fold-cv con una regressión logística

```
In [15]: # Leemos el dataset
         library(ISLR)
         names(Smarket)
         summary(Smarket)
```

1. 'Year' 2. 'Lag1' 3. 'Lag2' 4. 'Lag3' 5. 'Lag4' 6. 'Lag5' 7. 'Volume' 8. 'Today' 9. 'Direction'

```
      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
```

In [16]:
```
pairs(Smarket, col=Smarket$Direction)
```




```
In [17]: glmFit = train(Smarket[,-9],y= Smarket[,9],
                        method = "glm", preProcess = c("center","scale"),
                        tuneLength = 10,control=glm.control(maxit=500),
                        trControl = trainControl(method = "cv"))
         glmFit

Warning message:
"glm.fit: fitted probabilities numerically 0 or 1 occurred"Warning message:
"glm.fit: fitted probabilities numerically 0 or 1 occurred"Warning message:
"glm.fit: fitted probabilities numerically 0 or 1 occurred"Warning message:
"glm.fit: fitted probabilities numerically 0 or 1 occurred"Warning message:
"glm.fit: fitted probabilities numerically 0 or 1 occurred"Warning message:
"glm.fit: fitted probabilities numerically 0 or 1 occurred"Warning message:
```

```
"glm.fit: fitted probabilities numerically 0 or 1 occurred"Warning message:
"glm.fit: fitted probabilities numerically 0 or 1 occurred"Warning message:
"glm.fit: fitted probabilities numerically 0 or 1 occurred"Warning message:
"glm.fit: fitted probabilities numerically 0 or 1 occurred"Warning message:
"glm.fit: fitted probabilities numerically 0 or 1 occurred"

Generalized Linear Model

1250 samples
   8 predictor
   2 classes: 'Down', 'Up'

Pre-processing: centered (8), scaled (8)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 1125, 1125, 1124, 1124, 1125, 1125, ...
Resampling results:

  Accuracy   Kappa
  0.9936062  0.9871929
```

0.1.3 Ejercicio 3

Probar LDA con todas la variables. Comparar con Regresión Logística y LDA. Por último, probar con QDA, comparar los resultados y pintarlos.

In [18]:
```
# Primero comprobamos si la varianza de todas las variables es igual.
# Y que la distribución de las variables son normales.

library(MASS)
library(ISLR)
shapiro.test(Smarket$Lag1)
shapiro.test(Smarket$Lag2)
shapiro.test(Smarket$Lag3)
shapiro.test(Smarket$Lag4)
shapiro.test(Smarket$Lag5)

# Dado los resultados del test, sabemos que los datos no tienen una
# distribución normal. Esto hará que LDA no funcione bien.
```

```
Shapiro-Wilk normality test

data:  Smarket$Lag1
W = 0.97219, p-value = 8.889e-15
```

```
Shapiro-Wilk normality test

data:  Smarket$Lag2
W = 0.97217, p-value = 8.798e-15
```

```
Shapiro-Wilk normality test

data:  Smarket$Lag3
W = 0.9724, p-value = 1.035e-14
```

```
Shapiro-Wilk normality test

data:  Smarket$Lag4
W = 0.97242, p-value = 1.049e-14
```

```
Shapiro-Wilk normality test

data:  Smarket$Lag5
W = 0.97011, p-value = 2.149e-15
```

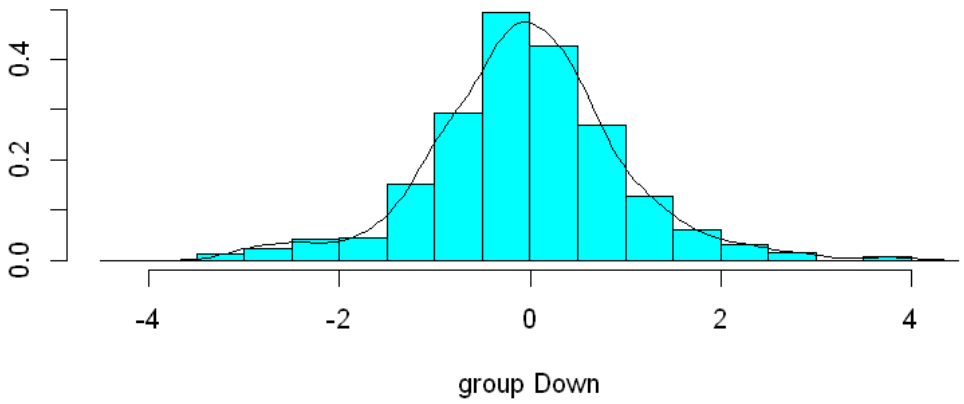
In [19]:
```
var(Smarket$Lag1)
var(Smarket$Lag2)
var(Smarket$Lag3)
var(Smarket$Lag4)
var(Smarket$Lag5)

# Como se puede ver, las varianzas de las variables son parecidas, menos para
# el caso de la variable 'Lag5', esto afectará a los resultados del LDA
```

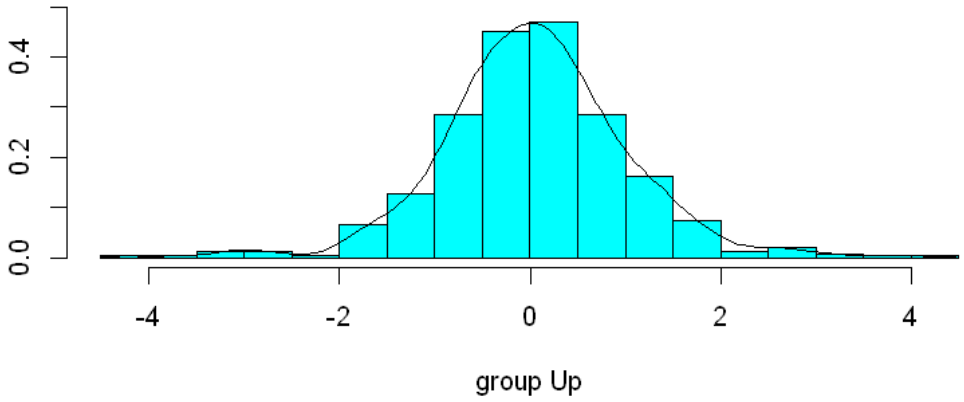
1.29117506222642
1.2911328189265
1.29664442448359
1.29680562160128
1.31687148397502

In [20]:
```
# Ahora aplicamos LDA.
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
```

```
In [21]: plot(lda.fit, type="both", xlab="LD1", ylab="Normalized frecuency")
```





In [22]:
```
# Ahora por ejemplo utilizaremos un subconjunto de los datos para
# comparar con Regressión Lineal.
Smarket.2005 = subset(Smarket,Year==2005)
glmFit = glm(Direction~Lag1+Lag2+Lag3+Lag4+Lag5,data=Smarket,family=binomial,
             subset=Year<2005)
glm.pred = predict(glmFit,Smarket.2005,type="response")
glm.pred = data.frame(prob=glm.pred)
glm.pred$class = ifelse(glm.pred$prob < 0.5, "Down","Up")
lda.pred = predict(lda.fit,Smarket.2005)

print("Tabla de resultados para glm")
table(glm.pred$class,Smarket.2005$Direction)
mean(glm.pred$class==Smarket.2005$Direction)
```

```
        print("Tabla de resultados para lda")
        table(lda.pred$class,Smarket.2005$Direction)
        mean(lda.pred$class==Smarket.2005$Direction)

        # Para estos datos y para las variables elegidas el error es el mismo.
        # Por lo cual ninguno de los dos es buen predictor.
```

```
[1] "Tabla de resultados para glm"
```

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

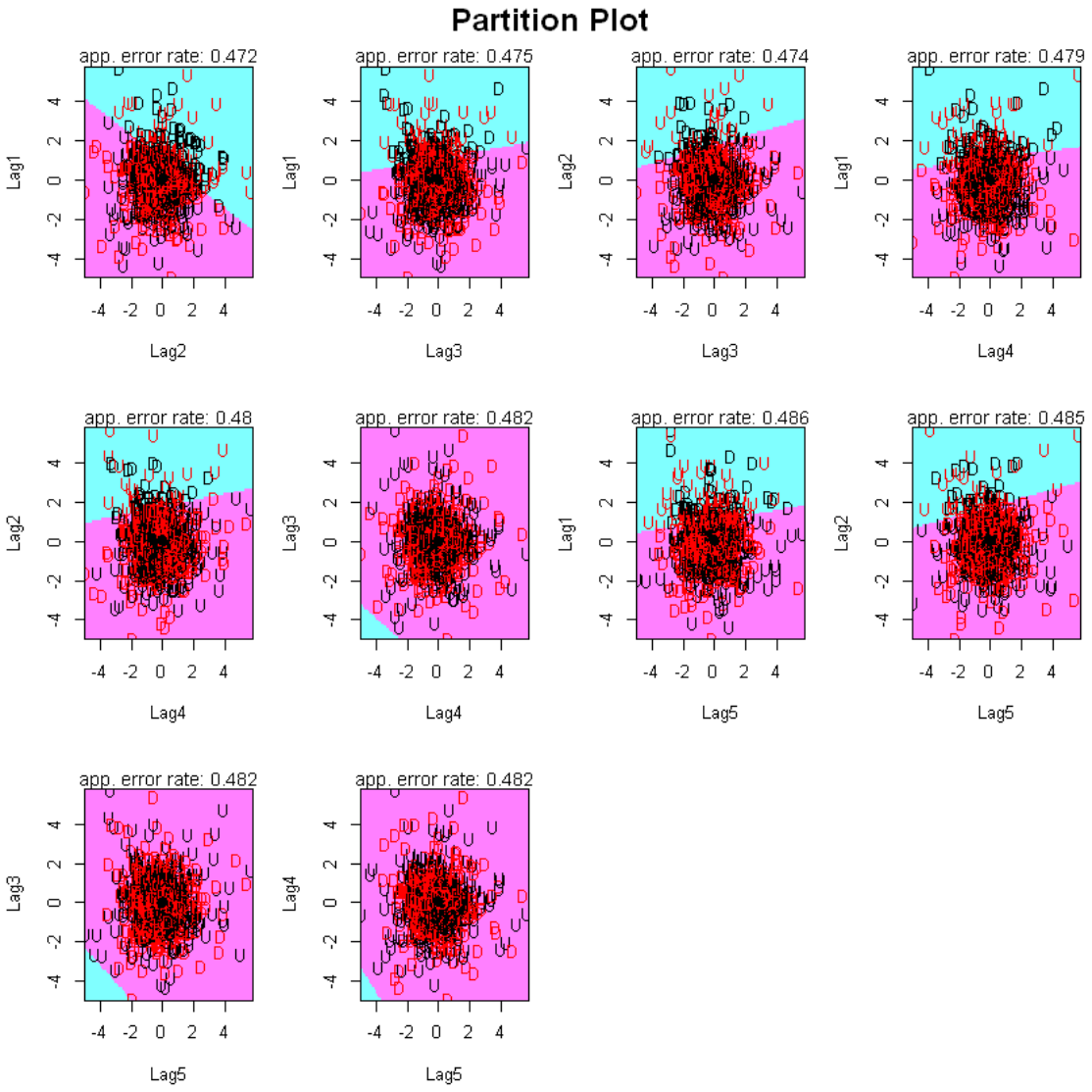
0.587301587301587

```
[1] "Tabla de resultados para lda"
```

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

0.587301587301587

```
In [23]: library(klaR)
         partimat(Direction~Lag1+Lag2+Lag3+Lag4+Lag5,data=Smarket, method="lda")
```



In [24]:
```
# Ahora pasamos a ejecutar QDA, pero primero debemos ver si las varianzas
# de los predictores para cada clase son diferentes.
var(Smarket[Smarket$Direction == "Up",]$Lag1)
var(Smarket[Smarket$Direction == "Up",]$Lag2)
var(Smarket[Smarket$Direction == "Down",]$Lag1)
var(Smarket[Smarket$Direction == "Down",]$Lag2)
var(Smarket[Smarket$Direction == "Up",]$Lag3)
var(Smarket[Smarket$Direction == "Up",]$Lag4)
var(Smarket[Smarket$Direction == "Down",]$Lag3)
var(Smarket[Smarket$Direction == "Down",]$Lag4)
var(Smarket[Smarket$Direction == "Up",]$Lag5)
var(Smarket[Smarket$Direction == "Down",]$Lag5)
```

1.27913706688038

1.24717074806801
1.30204143704291
1.33905195969342
1.27785074789389
1.22092377886303
1.31893272233984
1.38060525375758
1.36348271540061
1.26880333331491

In [25]:
```
# Ejecutamos el algoritmo de QDA.
qda.fit = qda(Direction~Lag1+Lag2+Lag3+Lag4+Lag5, data=Smarket,
              subset=Year<2005)
qda.fit
```

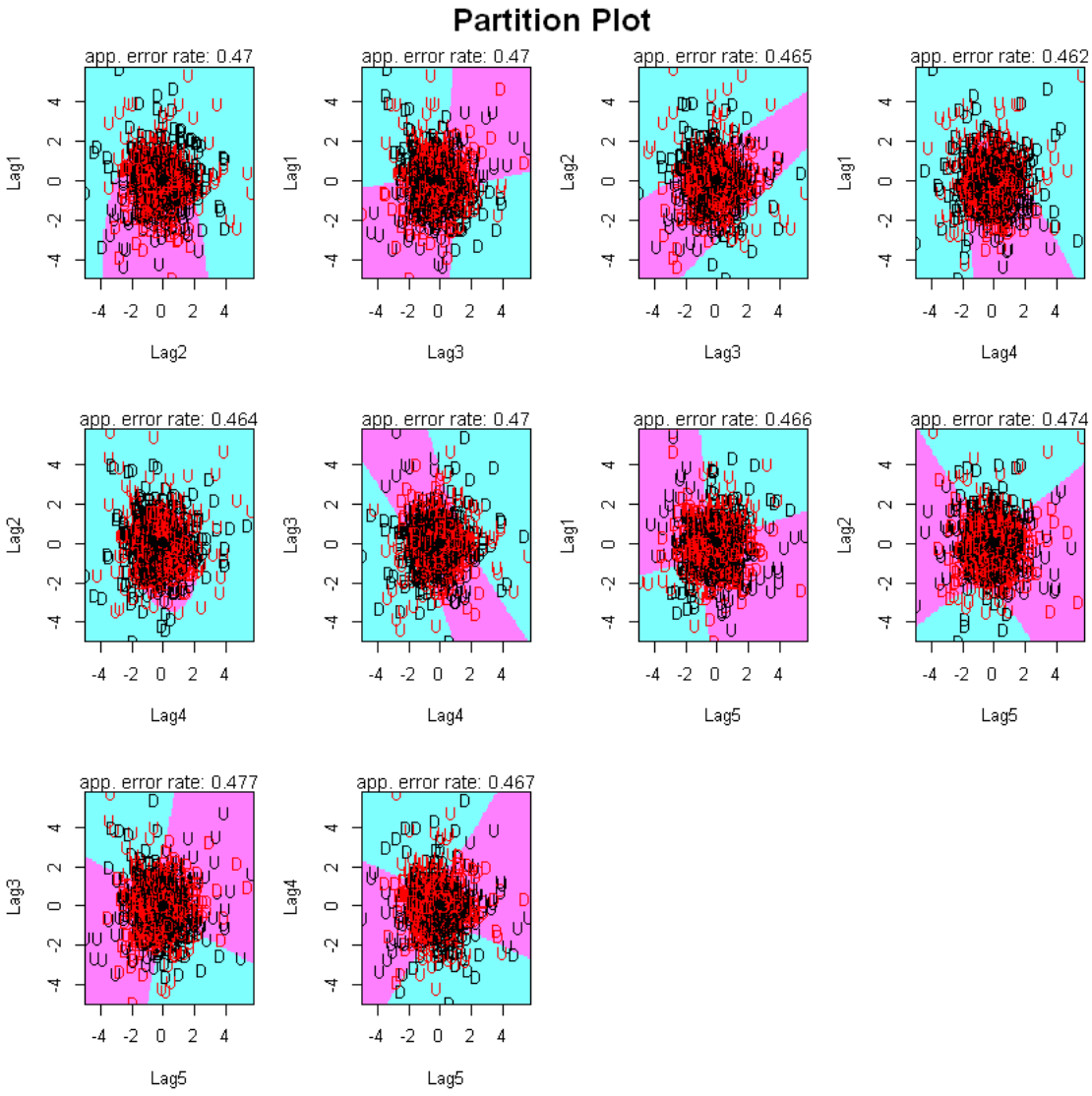
```
Call:
qda(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5, data = Smarket,
    subset = Year < 2005)

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
```

In [26]:
```
# Predecimos los datos para qda.
qda.pred = predict(qda.fit, Smarket.2005)
table(qda.pred$class, Smarket.2005$Direction)
mean(qda.pred$class==Smarket.2005$Direction)

# Como se puede ver, aunque  el resultado no sea mucho mejor que para LDA
# o para Regresión Lineal.
```

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

0.567460317460317

In [27]:
```
# Ahora mostrarermos los resultados de cada uno de los modelos.
partimat(Direction~Lag1+Lag2+Lag3+Lag4+Lag5,data=Smarket, method="qda")
```



0.1.4 Ejercicio 4

Usando la información del dataset 'clasif_train_alumnos.csv' * Comparar lda y qda usando Wilcoxon. * Hacer una comparación múltiple usando Friedman. * Usando Holm, mirar si hay algún algoritmo ganador.

```
In [28]: # Leemos el dataset.
         alum = read.csv('clasif_train_alumnos.csv',header=TRUE,stringsAsFactors=FALSE)
         head(alum)
         str(alum)
```

X	out_train_knn	out_train_lda	out_train_qda
appendicitis	0.8834602	0.8815461	0.8690241
australian	0.7277419	0.8605475	0.8072464
balance	0.9072122	0.8791122	0.9167999
bupa	0.7405521	0.7024224	0.6447628
contraceptive	0.6168944	0.5236485	0.5314180
haberman	0.7795116	0.7519934	0.7567115

```
'data.frame':        20 obs. of  4 variables:
 $ X            : chr  "appendicitis" "australian" "balance" "bupa" ...
 $ out_train_knn: num  0.883 0.728 0.907 0.741 0.617 ...
 $ out_train_lda: num  0.882 0.861 0.879 0.702 0.524 ...
 $ out_train_qda: num  0.869 0.807 0.917 0.645 0.531 ...
```

In [29]:
```
# Realizamos el test de wilcoxon.
QDAvsLDA = wilcox.test(alum$out_train_lda,alum$out_train_qda,
                       alternative="two.sided", paired=TRUE)
Rmas = QDAvsLDA$statistic
pvalue = QDAvsLDA$p.value
QDAvsLDA = wilcox.test(alum$out_train_qda,alum$out_train_lda,
                       alternative="two.sided",paired=TRUE)
Rmenos = QDAvsLDA$statistic

Rmas
Rmenos
pvalue


# Según el valor del p-value, podemos decir que los algoritmos son diferentes
# con un 82.31% ((1-pvalue)*100) de confianza.
```

V: 68
V: 142
0.176853179931641

In [30]:
```
# Realizamos el test de Friedman para todos los dataset.
test_friedman = friedman.test(as.matrix(alum))
test_friedman

# Según el resultado del test de friedman, podemos decir que al menos hay
# diferencias significativas entre un par de algortimos.
```

```
Friedman rank sum test

data:  as.matrix(alum)
Friedman chi-squared = 36.78, df = 3, p-value = 5.122e-08
```

In [36]:
```
# Por último, realizamos el test de Holm para comprobar si realmente hay algún
# algoritmo que se diferencia del resto.
tablatst = cbind(alum[,2:dim(alum)[2]])
rownames(tablatst) = alum[,1]
head(tablatst)
tam = dim(tablatst)
groups = as.numeric(rep(1:tam[2],each=tam[1]))
pairwise.wilcox.test(as.matrix(tablatst),groups,p.adjust="holm",paired=TRUE)

# Según los resultados obtenidos por el test de Holm, no hay diferencias significativ
# entre los diferentes algoritmos y por lo tanto no podemos concluir si un algoritmo
# es mejor que los demás.
```

	out_train_knn	out_train_lda	out_train_qda
appendicitis	0.8834602	0.8815461	0.8690241
australian	0.7277419	0.8605475	0.8072464
balance	0.9072122	0.8791122	0.9167999
bupa	0.7405521	0.7024224	0.6447628
contraceptive	0.6168944	0.5236485	0.5314180
haberman	0.7795116	0.7519934	0.7567115

1. 1 2. 1 3. 1 4. 1 5. 1 6. 1 7. 1 8. 1 9. 1 10. 1 11. 1 12. 1 13. 1 14. 1 15. 1 16. 1 17. 1 18. 1 19. 1 20. 1 21. 2 22. 2 23. 2 24. 2 25. 2 26. 2 27. 2 28. 2 29. 2 30. 2 31. 2 32. 2 33. 2 34. 2 35. 2 36. 2 37. 2 38. 2 39. 2 40. 2 41. 3 42. 3 43. 3 44. 3 45. 3 46. 3 47. 3 48. 3 49. 3 50. 3 51. 3 52. 3 53. 3 54. 3 55. 3 56. 3 57. 3 58. 3 59. 3 60. 3

```
Pairwise comparisons using Wilcoxon signed rank test

data:  as.matrix(tablatst) and groups

  1    2
2 0.65 -
3 0.59 0.53

P value adjustment method: holm
```