Clasificación Monótona

Francisco Pérez

15/2/2017

Lectura del Data set lev

Leemos el data set lev

```
library("foreign")
datos <- read.arff("lev.arff")
summary(datos)
```

```
##       In1             In2             In3             In4
##  Min.   :0.000   Min.   :0.000   Min.   :0.000   Min.   :0.000
##  1st Qu.:0.000   1st Qu.:1.000   1st Qu.:1.000   1st Qu.:1.000
##  Median :2.000   Median :2.000   Median :2.000   Median :2.000
##  Mean   :1.722   Mean   :1.985   Mean   :2.127   Mean   :1.985
##  3rd Qu.:3.000   3rd Qu.:3.000   3rd Qu.:3.000   3rd Qu.:3.000
##  Max.   :4.000   Max.   :4.000   Max.   :4.000   Max.   :4.000
##       Out1
##  Min.   :0.000
##  1st Qu.:1.000
##  Median :2.000
##  Mean   :1.785
##  3rd Qu.:2.000
##  Max.   :4.000
```

Veamos cuantas clases tiene este dataset

```
clases = as.integer(unique(datos$Out1))
clases
```

```
## [1] 3 2 0 4 1
```

```
length(clases)
```

```
## [1] 5
```

Teniendo un total de 5 clases Pasemos la clase a tipo factor:

```
datos$Out1 = as.factor(datos$Out1)
```

Trabajo con el Data set

Vamos a seleccionar los índices de las clases, una a una

```
indices.1.clase3 <- which(datos$Out1==clases[1])
indices.2.clase3y2 <- c(indices.1.clase3, which(datos$Out1==clases[2]))
indices.3.clases3y2y0 <- c(indices.2.clase3y2, which(datos$Out1==clases[3]))
indices.4.clases3y2y0y4 <- c(indices.3.clases3y2y0, which(datos$Out1==clases[4]))
```

Vamos a seleccionar a 0 solo la clase primera, y el resto a 1

```
p1 <- as.integer(datos$Out1)
p1[indices.1.clase3]<-0
p1 = ifelse(p1==0,0,1)
p2 <- as.integer(datos$Out1)
p2[indices.2.clase3y2]<-0
p2 = ifelse(p2==0,0,1)
p3 <- as.integer(datos$Out1)
p3[indices.3.clases3y2y0]<-0
p3 = ifelse(p3==0,0,1)
p4 <- as.integer(datos$Out1)
p4[indices.4.clases3y2y0y4]<-0
p4 = ifelse(p4==0,0,1)
```

Con lo que ya tenemos casi listo el primer data frame derivado, nos queda por juntar el resto del dataset con la nueva clase binaria.

```
data1 = cbind(datos[,1:4],target1=as.factor(p1))
data2 = cbind(datos[,1:4],target2=as.factor(p2))
data3 = cbind(datos[,1:4],target3=as.factor(p3))
data4 = cbind(datos[,1:4],target4=as.factor(p4))
```

Creación del modelo de clasificación

Vamos a usar el C4.5, implementado en el paquete de RWeka como J48.

```
library(RWeka)
```

```
##
## Attaching package: 'RWeka'
```

```
## The following objects are masked from 'package:foreign':
##
##     read.arff, write.arff
```

```
modelo1 <- J48(target1 ~., data = data1)
modelo1
```

```
## J48 pruned tree
## ------------------
##
## In2 <= 2
## |   In1 <= 3: 1 (542.0/16.0)
```

```
## |   In1 > 3
## |   |   In2 <= 1: 1 (89.0/11.0)
## |   |   In2 > 1
## |   |   |   In3 <= 3: 1 (16.0/5.0)
## |   |   |   In3 > 3: 0 (16.0/3.0)
## In2 > 2
## |   In1 <= 2
## |   |   In2 <= 3: 1 (80.0/20.0)
## |   |   In2 > 3
## |   |   |   In3 <= 2: 1 (95.0/34.0)
## |   |   |   In3 > 2: 0 (63.0/28.0)
## |   In1 > 2
## |   |   In1 <= 3: 0 (79.0/24.0)
## |   |   In1 > 3: 1 (20.0/8.0)
##
## Number of Leaves  :  9
##
## Size of the tree :   17
```

```
modelo2 <- J48(target2 ~., data = data2)
modelo2
```

```
## J48 pruned tree
## ------------------
##
## In2 <= 1
## |   In1 <= 1: 1 (157.0/16.0)
## |   In1 > 1
## |   |   In3 <= 2
## |   |   |   In4 <= 2: 1 (100.0/29.0)
## |   |   |   In4 > 2: 0 (41.0/16.0)
## |   |   In3 > 2
## |   |   |   In4 <= 2: 0 (42.0/9.0)
## |   |   |   In4 > 2: 1 (25.0/10.0)
## In2 > 1
## |   In1 <= 1
## |   |   In3 <= 1
## |   |   |   In2 <= 3
## |   |   |   |   In4 <= 3: 1 (78.0/16.0)
## |   |   |   |   In4 > 3: 0 (12.0/4.0)
## |   |   |   In2 > 3: 0 (16.0/2.0)
## |   |   In3 > 1
## |   |   |   In4 <= 0
## |   |   |   |   In1 <= 0: 0 (21.0/7.0)
## |   |   |   |   In1 > 0: 1 (7.0/2.0)
## |   |   |   In4 > 0: 0 (146.0/26.0)
## |   In1 > 1: 0 (355.0/42.0)
##
## Number of Leaves  :  12
##
## Size of the tree :   23
```

```
modelo3 <- J48(target3 ~., data = data3)
modelo3
```

```
## J48 pruned tree
## ------------------
##
## In2 <= 1
## |   In4 <= 1
## |   |   In1 <= 0: 0 (35.0/5.0)
## |   |   In1 > 0
## |   |   |   In1 <= 3
## |   |   |   |   In2 <= 0: 0 (33.0/12.0)
## |   |   |   |   In2 > 0: 1 (46.0/15.0)
## |   |   |   In1 > 3: 0 (42.0/7.0)
## |   In4 > 1
## |   |   In4 <= 2
## |   |   |   In1 <= 2: 1 (65.0/14.0)
## |   |   |   In1 > 2
## |   |   |   |   In1 <= 3: 0 (10.0/3.0)
## |   |   |   |   In1 > 3: 1 (23.0/8.0)
## |   |   In4 > 2: 0 (111.0/53.0)
## In2 > 1
## |   In1 <= 1
## |   |   In3 <= 1
## |   |   |   In2 <= 3
## |   |   |   |   In4 <= 3: 1 (78.0/25.0)
## |   |   |   |   In4 > 3: 0 (12.0/4.0)
## |   |   |   In2 > 3: 0 (16.0/1.0)
## |   |   In3 > 1: 0 (174.0/33.0)
## |   In1 > 1: 0 (355.0/39.0)
##
## Number of Leaves  :  13
##
## Size of the tree :   25
```

```
modelo4 <- J48(target4 ~., data = data4)
modelo4
```

```
## J48 pruned tree
## ------------------
##
## In2 <= 2
## |   In3 <= 2
## |   |   In4 <= 2
## |   |   |   In4 <= 1
## |   |   |   |   In2 <= 0
## |   |   |   |   |   In3 <= 1: 0 (14.0/3.0)
## |   |   |   |   |   In3 > 1: 1 (6.0/2.0)
## |   |   |   |   In2 > 0
## |   |   |   |   |   In3 <= 0: 1 (42.0/15.0)
## |   |   |   |   |   In3 > 0
## |   |   |   |   |   |   In1 <= 0: 0 (7.0)
## |   |   |   |   |   |   In1 > 0
```

```
## |   |   |   |   |   |   |   In2 <= 1: 1 (35.0/12.0)
## |   |   |   |   |   |   |   In2 > 1: 0 (16.0/7.0)
## |   |   |   In4 > 1
## |   |   |   |   In1 <= 2: 1 (76.0/20.0)
## |   |   |   |   In1 > 2
## |   |   |   |   |   In1 <= 3: 0 (16.0/3.0)
## |   |   |   |   |   In1 > 3: 1 (23.0/8.0)
## |   |   In4 > 2
## |   |   |   In1 <= 0: 1 (35.0/13.0)
## |   |   |   In1 > 0: 0 (118.0/25.0)
## |   In3 > 2
## |   |   In2 <= 1
## |   |   |   In1 <= 3
## |   |   |   |   In2 <= 0: 0 (68.0/22.0)
## |   |   |   |   In2 > 0: 1 (33.0/10.0)
## |   |   |   In1 > 3: 0 (48.0/8.0)
## |   |   In2 > 1
## |   |   |   In4 <= 1
## |   |   |   |   In1 <= 1: 1 (7.0/2.0)
## |   |   |   |   In1 > 1: 0 (26.0/5.0)
## |   |   |   In4 > 1: 0 (93.0/6.0)
## In2 > 2: 0 (337.0/26.0)
##
## Number of Leaves  :  18
##
## Size of the tree :   35
```

Hagamos un estudio más detallado de los modelos con la función “evaluate_Weka_classifier”:

```
evaluacion.modelo.1 <- evaluate_Weka_classifier(modelo1, numFolds = 10, complexity = FALSE, class = TRU
evaluacion.modelo.1
```

```
## === 10 Fold Cross Validation ===
##
## === Summary ===
##
## Correctly Classified Instances         833               83.3    %
## Incorrectly Classified Instances       167               16.7    %
## Kappa statistic                          0.4434
## Mean absolute error                      0.2182
## Root mean squared error                  0.3394
## Relative absolute error                 68.8852 %
## Root relative squared error             85.3208 %
## Total Number of Instances             1000
##
## === Detailed Accuracy By Class ===
##
##                  TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
##                  0,508    0,087    0,588      0,508    0,545      0,445    0,837     0,554     0
##                  0,913    0,492    0,883      0,913    0,898      0,445    0,837     0,943     1
## Weighted Avg.    0,833    0,413    0,825      0,833    0,828      0,445    0,837     0,867
##
## === Confusion Matrix ===
##
```

```
##    a   b   <-- classified as
##  100  97 |   a = 0
##   70 733 |   b = 1
```

```
evaluacion.modelo.2 <- evaluate_Weka_classifier(modelo1, numFolds = 10, complexity = FALSE, class = TRU
evaluacion.modelo.2
```

```
## === 10 Fold Cross Validation ===
##
## === Summary ===
##
## Correctly Classified Instances         834               83.4    %
## Incorrectly Classified Instances       166               16.6    %
## Kappa statistic                          0.4294
## Mean absolute error                      0.2158
## Root mean squared error                  0.336
## Relative absolute error                 68.1082 %
## Root relative squared error             84.4687 %
## Total Number of Instances             1000
##
## === Detailed Accuracy By Class ===
##
##                  TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
##                  0,472    0,077    0,600      0,472    0,528      0,434    0,844     0,560     0
##                  0,923    0,528    0,877      0,923    0,899      0,434    0,844     0,944     1
## Weighted Avg.    0,834    0,439    0,822      0,834    0,826      0,434    0,844     0,869
##
## === Confusion Matrix ===
##
##    a   b   <-- classified as
##   93 104 |   a = 0
##   62 741 |   b = 1
```

```
evaluacion.modelo.3 <- evaluate_Weka_classifier(modelo1, numFolds = 10, complexity = FALSE, class = TRU
evaluacion.modelo.3
```

```
## === 10 Fold Cross Validation ===
##
## === Summary ===
##
## Correctly Classified Instances         822               82.2    %
## Incorrectly Classified Instances       178               17.8    %
## Kappa statistic                          0.383
## Mean absolute error                      0.2226
## Root mean squared error                  0.3418
## Relative absolute error                 70.2812 %
## Root relative squared error             85.9326 %
## Total Number of Instances             1000
##
## === Detailed Accuracy By Class ===
##
##                  TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
##                  0,431    0,082    0,563      0,431    0,489      0,388    0,828     0,521     0
```

```
##                  0,918    0,569    0,868      0,918    0,892      0,388    0,828     0,938     1
## Weighted Avg.    0,822    0,473    0,808      0,822    0,813      0,388    0,828     0,855
##
## === Confusion Matrix ===
##
##    a   b   <-- classified as
##   85 112 |   a = 0
##   66 737 |   b = 1
```

```
evaluacion.modelo.4 <- evaluate_Weka_classifier(modelo1, numFolds = 10, complexity = FALSE, class = TRU
evaluacion.modelo.4
```

```
## === 10 Fold Cross Validation ===
##
## === Summary ===
##
## Correctly Classified Instances         818               81.8    %
## Incorrectly Classified Instances       182               18.2    %
## Kappa statistic                          0.3692
## Mean absolute error                      0.2259
## Root mean squared error                  0.3504
## Relative absolute error                 71.3206 %
## Root relative squared error             88.0949 %
## Total Number of Instances             1000
##
## === Detailed Accuracy By Class ===
##
##                  TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
##                  0,421    0,085    0,550      0,421    0,477      0,374    0,824     0,493     0
##                  0,915    0,579    0,866      0,915    0,890      0,374    0,824     0,937     1
## Weighted Avg.    0,818    0,481    0,803      0,818    0,809      0,374    0,824     0,850
##
## === Confusion Matrix ===
##
##    a   b   <-- classified as
##   83 114 |   a = 0
##   68 735 |   b = 1
```

Necesitamos conocer las probabilidades generadas por nuestros modelos, para ello probaremos a predecir la instancia 500 de nuestro dataset, sabiendo de por si que pertenece a la clase:

```
datos[500,3]
```

```
## [1] 2
```

```
prediccion1 <- predict(modelo1,datos[500,1:4],type="probability")
prediccion2 <- predict(modelo2,datos[500,1:4],type="probability")
prediccion3 <- predict(modelo3,datos[500,1:4],type="probability")
prediccion4 <- predict(modelo4,datos[500,1:4],type="probability")
```

Es decir, pertenece a la clase 2, por lo que en la primera predicción nos debería salir que es 1, y en el resto nos debería dar mayor probabilidad a la 0, ya que a partir de la predicción 2, la clase 2, estaba incluida en las particiones

```
prediccion1
```

```
##             0         1
## 500 0.0295203 0.9704797
```

```
prediccion2
```

```
##             0         1
## 500 0.6097561 0.3902439
```

```
prediccion3
```

```
##             0         1
## 500 0.5225225 0.4774775
```

```
prediccion4
```

```
##             0         1
## 500 0.7881356 0.2118644
```

Como se puede observar, ocurre como se comentaba, pero el comportamiento de la predicción 3 no es del todo bueno, al bajar la probabilidad.

Automatización del proceso

A continuación, vamos a realizar una serie de funciones para que este proceso se haga de forma automática con cualquier dataset. La primera será una función que realice la lectura del dataset y transforme la clase en factor y devuelva el número de clases que tiene el dataset.

```
LecturaYPreprocesado = function(fichero){
  datos <- read.arff(fichero)
  datos[dim(datos)[2]]
  clases = as.integer(unique(datos[[dim(datos)[2]]]))
  datos[[dim(datos)[2]]] = as.factor(datos[[dim(datos)[2]]])
  return(list(datos,clases))
}
salida.funcion.lectura <- LecturaYPreprocesado("lev.arff")
datos <- salida.funcion.lectura[[1]]
clases <- salida.funcion.lectura[[2]]
clases
```

```
## [1] 3 2 0 4 1
```

Ya tenemos la función que realiza el proceso de leer y preprocesar el conjunto de datos. Ahora vamos a seleccionar los índices de las clases, una a una, de forma automática con otra función

```
ObtenerIndices = function(fichero.datos,clases.totales){
  lista <- NULL
  for (i in 1:(length(clases.totales)-1)){
    if (i==1){
      indices <- which(fichero.datos[[dim(fichero.datos)[2]]]==clases.totales[i])
      lista <- list(indices)
    } else {
      indices <- c(lista[[length(lista)]], which(fichero.datos[[dim(fichero.datos)[2]]]==clases.totales
      lista <- c(lista, list(indices))
    }
  }
  return(lista)
}
lista.con.los.indices <- ObtenerIndices(datos,clases)
```

Una vez que tenemos todos los índices, vamos a seleccionar con 0 las clases primarias y el resto a 1, con otra función:

```
SeleccionDeClases = function(fichero.datos, clases.totales,lista){
  lista.salida <- NULL
  for (i in 1:(length(clases.totales)-1)){
    p <- as.integer(datos[[dim(datos)[2]]])
    p[lista[[i]]]<-0
    p <- ifelse(p==0,0,1)
    if (i==1){
      lista.salida <- list(p)
    } else {
      lista.salida <- c(lista.salida, list(p))
    }
  }
  return(lista.salida)
}
lista.con.la.seleccion.de.clases <- SeleccionDeClases(datos,clases,lista.con.los.indices)
```

Ahora juntaremos los dataset con las clases binarias, con otra función:

```
CreacionDataFramesBinarios = function(fichero.datos, clases.totales,lista.clases){
  lista.salida <- NULL
  for (i in 1:(length(clases.totales)-1)){
    data <- cbind(fichero.datos[,1:(dim(fichero.datos)[2]-1)],target=as.factor(lista.clases[[i]]))
    if (i==1){
      lista.salida <- list(data)
    } else {
      lista.salida <- c(lista.salida, list(data))
    }
  }
  return(lista.salida)
}
lista.data.frames.binarios <- CreacionDataFramesBinarios(datos,clases,lista.con.la.seleccion.de.clases)
```

El siguiente paso será crear los modelos usando el clasificador que queramos, primero probaremos con J48:

```
library(RWeka)
CreacionDeLosModelos = function(lista.data.frames,clases.totales){
  lista.salida <- NULL
  for (i in 1:(length(clases.totales)-1)){
    modelo <- J48(target ~., data = lista.data.frames[[i]])
    if (i==1){
      lista.salida <- list(modelo)
    } else {
      lista.salida <- c(lista.salida, list(modelo))
    }
  }
  return(lista.salida)
}
lista.modelos <- CreacionDeLosModelos(lista.data.frames.binarios, clases)
```

Vamos a realizar la evaluación de los modelos:

```
CrearEvaluacionDeModelos = function(lista.modelo, clases.totales){
  lista.salida <- NULL
  for (i in 1:(length(clases.totales)-1)){
    evaluacion <- evaluate_Weka_classifier(lista.modelo[[i]], numFolds = 10, complexity = FALSE, class =
    if (i==1){
      lista.salida <- list(evaluacion)
    } else {
      lista.salida <- c(lista.salida, list(evaluacion))
    }
  }
  return(lista.salida)
}
lista.evaluaciones <- CrearEvaluacionDeModelos(lista.modelos,clases)
```

Podemos ver la lista de las evaluaciones:

```
for(i in 1:(length(lista.evaluaciones))){
  print(lista.evaluaciones[[i]])
}
```

```
## === 10 Fold Cross Validation ===
##
## === Summary ===
##
## Correctly Classified Instances         782               78.2    %
## Incorrectly Classified Instances       218               21.8    %
## Kappa statistic                          0.4435
## Mean absolute error                      0.2978
## Root mean squared error                  0.3969
## Relative absolute error                 73.8192 %
## Root relative squared error             88.4016 %
## Total Number of Instances             1000
##
## === Detailed Accuracy By Class ===
##
```

```
##                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
##                 0,867    0,436    0,836      0,867    0,851      0,445    0,774     0,867     0
##                 0,564    0,133    0,622      0,564    0,592      0,445    0,774     0,546     1
## Weighted Avg.   0,782    0,351    0,776      0,782    0,779      0,445    0,774     0,777
##
## === Confusion Matrix ===
##
##    a   b   <-- classified as
##  624  96 |   a = 0
##  122 158 |   b = 1
## === 10 Fold Cross Validation ===
##
## === Summary ===
##
## Correctly Classified Instances         783               78.3    %
## Incorrectly Classified Instances       217               21.7    %
## Kappa statistic                          0.443
## Mean absolute error                      0.2982
## Root mean squared error                  0.3967
## Relative absolute error                 73.9113 %
## Root relative squared error             88.353  %
## Total Number of Instances             1000
##
## === Detailed Accuracy By Class ===
##
##                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
##                 0,871    0,443    0,835      0,871    0,852      0,444    0,771     0,866     0
##                 0,557    0,129    0,627      0,557    0,590      0,444    0,771     0,525     1
## Weighted Avg.   0,783    0,355    0,777      0,783    0,779      0,444    0,771     0,770
##
## === Confusion Matrix ===
##
##    a   b   <-- classified as
##  627  93 |   a = 0
##  124 156 |   b = 1
## === 10 Fold Cross Validation ===
##
## === Summary ===
##
## Correctly Classified Instances         793               79.3    %
## Incorrectly Classified Instances       207               20.7    %
## Kappa statistic                          0.4769
## Mean absolute error                      0.2943
## Root mean squared error                  0.3945
## Relative absolute error                 72.9496 %
## Root relative squared error             87.8658 %
## Total Number of Instances             1000
##
## === Detailed Accuracy By Class ===
##
##                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
##                 0,868    0,400    0,848      0,868    0,858      0,477    0,770     0,863     0
##                 0,600    0,132    0,639      0,600    0,619      0,477    0,770     0,537     1
## Weighted Avg.   0,793    0,325    0,789      0,793    0,791      0,477    0,770     0,772
```

```
##
## === Confusion Matrix ===
##
##    a   b   <-- classified as
##  625  95 |   a = 0
##  112 168 |   b = 1
## === 10 Fold Cross Validation ===
##
## === Summary ===
##
## Correctly Classified Instances         792               79.2    %
## Incorrectly Classified Instances       208               20.8    %
## Kappa statistic                          0.4726
## Mean absolute error                      0.293
## Root mean squared error                  0.3929
## Relative absolute error                 72.6256 %
## Root relative squared error             87.4975 %
## Total Number of Instances             1000
##
## === Detailed Accuracy By Class ===
##
##                  TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
##                  0,869    0,407    0,846      0,869    0,858      0,473    0,783     0,883     0
##                  0,593    0,131    0,638      0,593    0,615      0,473    0,783     0,538     1
## Weighted Avg.    0,792    0,330    0,788      0,792    0,790      0,473    0,783     0,786
##
## === Confusion Matrix ===
##
##    a   b   <-- classified as
##  626  94 |   a = 0
##  114 166 |   b = 1
```

Una vez que tenemos todo el proceso automatizado, pasaremos a realizar las prediciones:

```
RealizarPrediccion = function(lista.modelo, fichero.datos, elemento){
  lista.salida <- list(fichero.datos[elemento,(dim(fichero.datos)[2])])
  for (i in 1:(length(lista.modelo))){
    prediccion <- predict(lista.modelo[[i]],fichero.datos[elemento,1:(dim(fichero.datos)[2]-1)],type="p
    lista.salida <- c(lista.salida, list(prediccion))
  }
  return(lista.salida)
}
elemento.elegido = 500
lista.predicciones <- RealizarPrediccion(lista.modelos, datos, elemento.elegido)
lista.predicciones
```

```
## [[1]]
## [1] 2
## Levels: 0 1 2 3 4
##
## [[2]]
##             0         1
## 500 0.0295203 0.9704797
```

```
##
## [[3]]
##             0         1
## 500 0.6097561 0.3902439
##
## [[4]]
##             0         1
## 500 0.5225225 0.4774775
##
## [[5]]
##             0         1
## 500 0.7881356 0.2118644
```

Y por último una función que explique la predicción:

```
ExplicarPrediccion = function(lista.prediccion, elemento, clases.totales){
  print("Tenemos que el elemento")
  print(elemento)
  print("Pertenece a la clase")
  print(lista.prediccion[[1]])
  clase.a.la.que.pertenece <- lista.prediccion[[1]]
  pertenece.al.0 <- FALSE
  predicciones.buenas = 0
  predicciones.malas = 0
  for (i in 2:(length(lista.prediccion))){
    print("Siendo la prediccion")
    print(lista.prediccion[[i]])
    print("Y las clases que pertenecen al 0: ")
    for (j in 1:(i-1)){
      print(clases.totales[j])
      if (clase.a.la.que.pertenece == clases.totales[j])
        pertenece.al.0 = TRUE
    }
    if (pertenece.al.0 & (lista.prediccion[[i]][1] > 0.5)){
      print("Por lo que pertenece a 0 y se hace una buena predicción al ser superior al 0.5")
      predicciones.buenas = predicciones.buenas + 1
    } else if (pertenece.al.0){
      print("No se realiza una buena predicción al pertenecer a 0 y no ser la predicción superior al 0.5
      predicciones.malas = predicciones.malas + 1
    } else if (lista.prediccion[[i]][2] > 0.5){
      print("Se hace una buena predicción al pertenecer a 1 y ser superior a 0.5")
      predicciones.buenas = predicciones.buenas + 1
    } else {
      print("No se hace una buena predicción al pertenecer a 1 y no ser superior a 0.5")
      predicciones.malas = predicciones.malas + 1
    }
  }
  print("En total tenemos ")
  print(predicciones.buenas)
  print("predicciones buenas y ")
  print(predicciones.malas)
  print("predicciones malas.")
}
ExplicarPrediccion(lista.predicciones, elemento.elegido, clases)
```

```
## [1] "Tenemos que el elemento"
## [1] 500
## [1] "Pertenece a la clase"
## [1] 2
## Levels: 0 1 2 3 4
## [1] "Siendo la prediccion"
##             0         1
## 500 0.0295203 0.9704797
## [1] "Y las clases que pertenecen al 0: "
## [1] 3
## [1] "Se hace una buena predicción al pertenecer a 1 y ser superior a 0.5"
## [1] "Siendo la prediccion"
##             0         1
## 500 0.6097561 0.3902439
## [1] "Y las clases que pertenecen al 0: "
## [1] 3
## [1] 2
## [1] "Por lo que pertenece a 0 y se hace una buena predicción al ser superior al 0.5"
## [1] "Siendo la prediccion"
##             0         1
## 500 0.5225225 0.4774775
## [1] "Y las clases que pertenecen al 0: "
## [1] 3
## [1] 2
## [1] 0
## [1] "Por lo que pertenece a 0 y se hace una buena predicción al ser superior al 0.5"
## [1] "Siendo la prediccion"
##             0         1
## 500 0.7881356 0.2118644
## [1] "Y las clases que pertenecen al 0: "
## [1] 3
## [1] 2
## [1] 0
## [1] 4
## [1] "Por lo que pertenece a 0 y se hace una buena predicción al ser superior al 0.5"
## [1] "En total tenemos "
## [1] 4
## [1] "predicciones buenas y "
## [1] 0
## [1] "predicciones malas."
```

Prueba con otro dataset

Vamos a probar todas las funciones anteriores con el dataset ESL:

```
salida.funcion.lectura <- LecturaYPreprocesado("esl.arff")
datos <- salida.funcion.lectura[[1]]
clases <- salida.funcion.lectura[[2]]
clases
```

```
## [1] 6 5 4 3 2 7 8 1 9
```

```
lista.con.los.indices <- ObtenerIndices(datos,clases)
lista.con.la.seleccion.de.clases <- SeleccionDeClases(datos,clases,lista.con.los.indices)
lista.data.frames.binarios <- CreacionDataFramesBinarios(datos,clases,lista.con.la.seleccion.de.clases)
library(party)
```

```
## Warning: package 'party' was built under R version 3.3.2
```

```
## Loading required package: grid
```

```
## Loading required package: mvtnorm
```

```
## Loading required package: modeltools
```

```
## Loading required package: stats4
```

```
## Loading required package: strucchange
```

```
## Loading required package: zoo
```

```
## Warning: package 'zoo' was built under R version 3.3.2
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
## Loading required package: sandwich
```

```
lista.modelos <- CreacionDeLosModelos(lista.data.frames.binarios, clases)
lista.evaluaciones <- CrearEvaluacionDeModelos(lista.modelos,clases)
elemento.elegido = 300
lista.predicciones <- RealizarPrediccion(lista.modelos, datos, elemento.elegido)
lista.predicciones
```

```
## [[1]]
## [1] 3
## Levels: 1 2 3 4 5 6 7 8 9
##
## [[2]]
##     0 1
## 300 0 1
##
## [[3]]
##               0         1
## 300 0.05714286 0.9428571
##
## [[4]]
##             0          1
```

```
## 300 0.9315068 0.06849315
##
## [[5]]
##             0          1
## 300 0.9549296 0.04507042
##
## [[6]]
##            0          1
## 300 0.980057 0.01994302
##
## [[7]]
##             0          1
## 300 0.9810427 0.01895735
##
## [[8]]
##             0          1
## 300 0.9877049 0.01229508
##
## [[9]]
##             0           1
## 300 0.9918033 0.008196721
```

```
ExplicarPrediccion(lista.predicciones, elemento.elegido, clases)
```

```
## [1] "Tenemos que el elemento"
## [1] 300
## [1] "Pertenece a la clase"
## [1] 3
## Levels: 1 2 3 4 5 6 7 8 9
## [1] "Siendo la prediccion"
##     0 1
## 300 0 1
## [1] "Y las clases que pertenecen al 0: "
## [1] 6
## [1] "Se hace una buena predicción al pertenecer a 1 y ser superior a 0.5"
## [1] "Siendo la prediccion"
##              0         1
## 300 0.05714286 0.9428571
## [1] "Y las clases que pertenecen al 0: "
## [1] 6
## [1] 5
## [1] "Se hace una buena predicción al pertenecer a 1 y ser superior a 0.5"
## [1] "Siendo la prediccion"
##             0          1
## 300 0.9315068 0.06849315
## [1] "Y las clases que pertenecen al 0: "
## [1] 6
## [1] 5
## [1] 4
## [1] "No se hace una buena predicción al pertenecer a 1 y no ser superior a 0.5"
## [1] "Siendo la prediccion"
##             0          1
## 300 0.9549296 0.04507042
## [1] "Y las clases que pertenecen al 0: "
```

```
## [1] 6
## [1] 5
## [1] 4
## [1] 3
## [1] "Por lo que pertenece a 0 y se hace una buena predicción al ser superior al 0.5"
## [1] "Siendo la prediccion"
##           0          1
## 300 0.980057 0.01994302
## [1] "Y las clases que pertenecen al 0: "
## [1] 6
## [1] 5
## [1] 4
## [1] 3
## [1] 2
## [1] "Por lo que pertenece a 0 y se hace una buena predicción al ser superior al 0.5"
## [1] "Siendo la prediccion"
##             0          1
## 300 0.9810427 0.01895735
## [1] "Y las clases que pertenecen al 0: "
## [1] 6
## [1] 5
## [1] 4
## [1] 3
## [1] 2
## [1] 7
## [1] "Por lo que pertenece a 0 y se hace una buena predicción al ser superior al 0.5"
## [1] "Siendo la prediccion"
##             0          1
## 300 0.9877049 0.01229508
## [1] "Y las clases que pertenecen al 0: "
## [1] 6
## [1] 5
## [1] 4
## [1] 3
## [1] 2
## [1] 7
## [1] 8
## [1] "Por lo que pertenece a 0 y se hace una buena predicción al ser superior al 0.5"
## [1] "Siendo la prediccion"
##             0           1
## 300 0.9918033 0.008196721
## [1] "Y las clases que pertenecen al 0: "
## [1] 6
## [1] 5
## [1] 4
## [1] 3
## [1] 2
## [1] 7
## [1] 8
## [1] 1
## [1] "Por lo que pertenece a 0 y se hace una buena predicción al ser superior al 0.5"
## [1] "En total tenemos "
## [1] 7
## [1] "predicciones buenas y "
```

```
## [1] 1
## [1] "predicciones malas."
```