Minería de Datos - Tarea 1

Francisco Alonso

February 13, 2016

Tarea 1

Ejercicio 1

La certificación de la calidad vino es un proceso que consume tiempo y puede resultar costoso, especialmente si se requiere de la evaluación realizada por expertos humanos. Con el fin de apoyar este proceso, se quiere aplicar la minería de datos para construir una aplicación que permita estimar la calidad de una muestra de vino a partir de variables fisicoquímicas.

Se aplicó el proceso de minería de datos usando el algoritmo de clasificación C5.0 que extiende el C4.5.

Luego de cargar la data se convierte la clase de valor numérico a nominal o "factor".

```
wine <- as.data.frame(read.csv("winequality-red.csv", sep = ";"))
wine$quality <- as.factor(wine$quality)
```

Se asigna la semilla para fines de reproducibilidad y se separa aleatoriamente el data set en un set de entrenamiento y uno de prueba.

```
set.seed(42)
trainingIndex <- sample(seq_len(nrow(wine)), size = (nrow(wine) * 0.8))

trainWine <- wine[trainingIndex, ]
testWine <- wine[-trainingIndex, ]
```

Se crea el modelo usando el set de entrenamiento especificando que el atributo a predecir es "quality". El parámetro "trials" especifica el número de iteraciones del algoritmo de aprendizaje intentando mejorar su precisión a partir de previas iteraciones.

```
wineTree <- C5.0(trainWine$quality~., data = trainWine, trials = 7)
```

La salida del algoritmo provee un arbol de decisión que para el set de entrenamiento tiene una tasa de errores de 0% luego del proceso de boosting. También se muestra la importancia de cada atributo para el modelo construido.

```
## Evaluation on training data (1279 cases):
##
## Trial        Decision Tree
## -----      ----------------
##    Size      Errors
##
##    0    192  116( 9.1%)
##    1    132  260(20.3%)
##    2    145  245(19.2%)
##    3    159  228(17.8%)
```

```
##    4    141  290(22.7%)
##    5    130  295(23.1%)
##    6    155  223(17.4%)
## boost              0( 0.0%)   <<
##
##
##     (a)   (b)   (c)   (d)   (e)   (f)    <-classified as
##    ----  ----  ----  ----  ----  ----
##       9                                   (a): class 3
##            38                             (b): class 4
##                 552                       (c): class 5
##                       511                 (d): class 6
##                             154           (e): class 7
##                                   15      (f): class 8
##
##
##  Attribute usage:
##
##  100.00% volatile.acidity
##  100.00% citric.acid
##  100.00% sulphates
##  100.00% alcohol
##   99.45% total.sulfur.dioxide
##   98.98% chlorides
##   98.36% fixed.acidity
##   98.12% residual.sugar
##   95.54% free.sulfur.dioxide
##   94.76% density
##   94.21% pH
##
##
## Time: 0.3 secs
```

Se prueba el modelo usando el set de prueba y construyendo una matriz de confusión.

```
prediction <- predict(wineTree, testWine)
table(testWine$quality, prediction)
```

```
##    prediction
##      3  4  5  6  7  8
##   3  0  0  1  0  0  0
##   4  0  0 11  3  1  0
##   5  0  1 91 35  2  0
##   6  0  2 17 99  8  1
##   7  0  0  6 20 18  1
##   8  0  0  0  1  2  0
```

Se observa que a pesar del proceso de boosting aún se producen clasificaciones erroneas. Especialmente entre las clases 6 y 7, un número considerable de elementos de la clase 7 fueron clasificados como clase 6.

Ejercicio 2

Para realizar estudios criminológicos sería conveniente disponer de una aplicación que, de manera automática, realice la identificación del tipo de vidrio en una escena de crimen. Como se dispone de un conjunto de datos

de muestras que han sido clasificadas por los expertos, se quiere utilizar la minería de datos para construir un primer prototipo de esta aplicación.

Para el proceso de minería de datos se usó el algoritmo C5.0. El primer paso es preparar la data. Se elimnina el atributo identificador, se le asigna un nombre a cada atributo para mejor indentificación y se cambia el tipo de dato de la clase a nominal ("factor").

```
glass <- as.data.frame(read.table("glass.data", sep = ","))
glass <- select(glass, -V1)
names(glass) <- c("RI", "Na", "Mg", "Al", "Si", "K", "Ca", "Ba", "Fe", "Type")
glass$Type <- as.factor(glass$Type)

set.seed(42)
trainingIndex <- sample(seq_len(nrow(glass)), size = (nrow(glass) * 0.8))

trainGlass <- glass[trainingIndex, ]
testGlass <- glass[-trainingIndex, ]
```

Luego de dividir aleatoriamente la data en conjunto de entrenamiento y conjunto de prueba, se procede a construir el arbol de decisión usando el algoritmo C5.0. Se asigna "trials = 5" luego de varios intentos para el proceso de boosting.

```
glassTree <- C5.0(trainGlass$Type~., data = trainGlass, trials = 5)
```

Por último se prueba el modelo en el conjunto de prueba y se extraen las métricas de evaluación.

```
prediction <- predict.C5.0(glassTree, testGlass)
table(True = testGlass$Type, predicted = prediction)
```

```
##     predicted
## True  1  2  3  5  6  7
##    1 14  2  0  0  0  0
##    2  3 12  0  0  0  0
##    3  2  0  0  0  0  0
##    5  0  1  0  0  0  1
##    6  0  2  0  0  1  1
##    7  0  0  0  1  0  3
```

```
confMatrix <- confusionMatrix(prediction, testGlass$Type)
confMatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  5  6  7
##          1 14  3  2  0  0  0
##          2  2 12  0  1  2  0
##          3  0  0  0  0  0  0
##          5  0  0  0  0  0  1
##          6  0  0  0  0  1  0
##          7  0  0  0  1  1  3
##
```

```
## Overall Statistics
##
##                Accuracy : 0.6977
##                  95% CI : (0.5387, 0.8282)
##     No Information Rate : 0.3721
##     P-Value [Acc > NIR] : 1.496e-05
##
##                   Kappa : 0.5578
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 5 Class: 6 Class: 7
## Sensitivity            0.8750   0.8000  0.00000  0.00000  0.25000  0.75000
## Specificity            0.8148   0.8214  1.00000  0.97561  1.00000  0.94872
## Pos Pred Value         0.7368   0.7059      NaN  0.00000  1.00000  0.60000
## Neg Pred Value         0.9167   0.8846  0.95349  0.95238  0.92857  0.97368
## Prevalence             0.3721   0.3488  0.04651  0.04651  0.09302  0.09302
## Detection Rate         0.3256   0.2791  0.00000  0.00000  0.02326  0.06977
## Detection Prevalence   0.4419   0.3953  0.00000  0.02326  0.02326  0.11628
## Balanced Accuracy      0.8449   0.8107  0.50000  0.48780  0.62500  0.84936
```

La exactitud arrojada es 0.6977. La media geométrica es calculada a continuación.

```
a <- confMatrix$byClass[,1]
mean.geometric(a)
```

```
## [1] 0.3954158
```

Ejercicio 3

Uno de los peligros inherentes a la actividad minera es la amenaza sísmica que ocurre con frecuencia en muchas minas subterráneas. Los factores que influyen en la naturaleza de estos eventos son muy diversos, y las relaciones entre estos factores son muy complejas y muy poco conocidas. Los métodos utilizados hasta ahora para anticipar la actividad sísmica peligrosa no cubren las necesidades en este sector, ya que resultan insuficientes para lograr una buena sensibilidad y especificidad en las predicciones. Es por esto que se ha planteado verificar si los métodos de la minería de datos pueden ser capaces de predecir eventos sísmicos peligrosos.

Se aplica el proceso de minería de datos usando los algoritmos C5.0 y RIPPER comparándolos en base a sensibldad y media geométrica.

```
seismic <- as.data.frame(read.arff("seismic-bumps.arff"))
seismic$class <- as.factor(seismic$class)

set.seed(42)
trainingIndex <- sample(seq_len(nrow(seismic)), size = (nrow(seismic) * 0.8))
trainSeismic <- seismic[trainingIndex, ]
testSeismic <- seismic[-trainingIndex, ]
```

Luego de preparar la data se genera el modelo sando C5.0.

```
seismicTreeC50 <- C5.0(trainSeismic$class~., data = trainSeismic)
predictionC50 <- predict.C5.0(seismicTreeC50, testSeismic)
print(confusionMatrix(predictionC50, testSeismic$class))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 493  24
##          1   0   0
##
##                Accuracy : 0.9536
##                  95% CI : (0.9317, 0.97)
##     No Information Rate : 0.9536
##     P-Value [Acc > NIR] : 0.554
##
##                   Kappa : 0
##  Mcnemar's Test P-Value : 2.668e-06
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9536
##          Neg Pred Value :    NaN
##              Prevalence : 0.9536
##          Detection Rate : 0.9536
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##
```

Se construye el modelo usando el algoritmo RIPPER.

```
seismicRIPPER <- RWeka::JRip(trainSeismic$class~., data = trainSeismic)
predictionRIPPER <- predict(seismicRIPPER, testSeismic)
print(confusionMatrix(predictionRIPPER, testSeismic$class))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 478  19
##          1  15   5
##
##                Accuracy : 0.9342
##                  95% CI : (0.9093, 0.954)
##     No Information Rate : 0.9536
##     P-Value [Acc > NIR] : 0.9819
##
##                   Kappa : 0.1932
##  Mcnemar's Test P-Value : 0.6069
##
```

```
##             Sensitivity : 0.9696
##             Specificity : 0.2083
##          Pos Pred Value : 0.9618
##          Neg Pred Value : 0.2500
##              Prevalence : 0.9536
##          Detection Rate : 0.9246
##    Detection Prevalence : 0.9613
##       Balanced Accuracy : 0.5890
##
##        'Positive' Class : 0
##
```

Los modelos arrojan una precision alta de 0.9536 y 0.9342 sin embargo la muestra está sesgada ya que no hay una cantidad representativa de la clase 1.

```
sampleClass0 <- filter(seismic, class == '0')
sampleClass1 <- filter(seismic, class == '1')
print("Class 0 rows ")
```

```
## [1] "Class 0 rows "
```

```
print(nrow(sampleClass0))
```

```
## [1] 2414
```

```
print("Class 1 rows ")
```

```
## [1] "Class 1 rows "
```

```
print(nrow(sampleClass1))
```

```
## [1] 170
```

Se corre el algoritmo SMOTE sobre el conjunto de datos para generar una mestra que contenga más filas de la clase 1.

```
seismicSMOTED <- SMOTE(class ~ ., data = seismic, perc.over = 500, k = 5)
trainingIndex <- sample(seq_len(nrow(seismicSMOTED)), size = (nrow(seismicSMOTED) * 0.8))
trainSeismic <- seismicSMOTED[trainingIndex, ]
testSeismic <- seismicSMOTED[-trainingIndex, ]
```

Luego se construyen los modelos nuevamente y se prueban con los nuevos conjuntos de prueba. Cabe destacar que en este caso el conjunto de datos permitió al algoritmo realizar el proceso de boosting.

- C5.0:

```
seismicTreeC50 <- C5.0(trainSeismic$class~., data = trainSeismic, trials = 20)
predictionC50 <- predict.C5.0(seismicTreeC50, testSeismic)
print(confusionMatrix(predictionC50, testSeismic$class))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 325  29
##          1  11 179
##
##                Accuracy : 0.9265
##                  95% CI : (0.9012, 0.947)
##     No Information Rate : 0.6176
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8417
##  Mcnemar's Test P-Value : 0.00719
##
##             Sensitivity : 0.9673
##             Specificity : 0.8606
##          Pos Pred Value : 0.9181
##          Neg Pred Value : 0.9421
##              Prevalence : 0.6176
##          Detection Rate : 0.5974
##    Detection Prevalence : 0.6507
##       Balanced Accuracy : 0.9139
##
##        'Positive' Class : 0
##
```

- RIPPER

```
seismicRIPPER <- RWeka::JRip(trainSeismic$class~., data = trainSeismic)
predictionRIPPER <- predict(seismicRIPPER, testSeismic)
print(confusionMatrix(predictionRIPPER, testSeismic$class))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 324  38
##          1  12 170
##
##                Accuracy : 0.9081
##                  95% CI : (0.8806, 0.931)
##     No Information Rate : 0.6176
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8007
##  Mcnemar's Test P-Value : 0.000407
##
##             Sensitivity : 0.9643
##             Specificity : 0.8173
##          Pos Pred Value : 0.8950
##          Neg Pred Value : 0.9341
##              Prevalence : 0.6176
```

```
##          Detection Rate : 0.5956
##    Detection Prevalence : 0.6654
##       Balanced Accuracy : 0.8908
##
##        'Positive' Class : 0
##
```

Ambos modelos presentan un buen desempeño, sin embargo, el algoritmo C5.0 supera al algoritmo RIPPER por un pequeño margen en cuanto a precisión y sensitividad.