# Unidad 4 - Módulo 1

*Pablo Anicich*

*24 de julio de 2016*

**Ejercicio árbol 3 resuelto**

Cargo todos los paquetes necesarios para trabajar con árboles:

```
library(rpart)
suppressWarnings(library(rattle))
```

```
## Rattle: A free graphical interface for data mining with R.
## Versión 4.1.0 Copyright (c) 2006-2015 Togaware Pty Ltd.
## Escriba 'rattle()' para agitar, sacudir y  rotar sus datos.
```

```
suppressWarnings(library(rpart.plot))
suppressWarnings(library(RColorBrewer))
```

Lo que primero que hago es setear el directorio de trabajo para leer el archivo **Arbol3.csv**:

```
setwd("C:/Documentos/Diplomatura_en_BI_UTN/Modulo_1/Unidad_4/");
arbol3 <- read.csv("Arbol3.csv",
                   header=TRUE, sep=";", na.strings=c("NA"));
```

Para familiarme con los datos, extraigo la info sobre la estructura de los datos en el dataframe **arbol3**.

```
str(arbol3)
```

```
## 'data.frame':    40000 obs. of  7 variables:
##  $ Atributo.1: int  1 0 0 0 0 0 0 0 0 0 ...
##  $ Atributo.2: int  0 0 1 0 0 0 0 0 1 0 ...
##  $ Atributo.3: int  1 0 1 1 1 0 1 0 1 1 ...
##  $ Atributo.4: int  0 0 1 0 1 0 1 1 1 0 ...
##  $ Atributo.5: int  1 1 1 0 1 1 1 1 1 0 ...
##  $ Atributo.6: int  0 1 1 1 1 1 1 1 1 1 ...
##  $ Resultado : int  1 0 0 0 0 0 0 0 0 0 ...
```

Son 7 variables con 40000 observaciones. Todas las variables son enteras.

**El ejercicio plantea, para el dataframe "arbol3", resolver lo siguiente:**

- a-Seleccionar tres conjuntos de datos al azar (a,b,c) a partir de la tabla arbol3.
- b-Construir una árbol de decisión a partir del subconjunto a.
- c-Realizar una poda usando el subconjunto b.
- d-Verificar que poder predictivo tiene cada nodo usando el subconjunto c.

**Solución:**

- **a**. Construyo los siguientes dataframes:

```
arbol3.train <- arbol3[1:15000,]     # diseño del árbol de decisión
arbol3.prune <- arbol3[15001:30000,] # para la poda
arbol3.test <- arbol3[30001:40000,]  # cross-validación del árbol
```

Y verifico los datasets obtenidos:

```
str(arbol3.train)
```

```
## 'data.frame':    15000 obs. of  7 variables:
##  $ Atributo.1: int  1 0 0 0 0 0 0 0 0 0 ...
##  $ Atributo.2: int  0 0 1 0 0 0 0 0 1 0 ...
##  $ Atributo.3: int  1 0 1 1 1 0 1 0 1 1 ...
##  $ Atributo.4: int  0 0 1 0 1 0 1 1 1 0 ...
##  $ Atributo.5: int  1 1 1 0 1 1 1 1 1 0 ...
##  $ Atributo.6: int  0 1 1 1 1 1 1 1 1 1 ...
##  $ Resultado : int  1 0 0 0 0 0 0 0 0 0 ...
```

```
str(arbol3.prune)
```

```
## 'data.frame':    15000 obs. of  7 variables:
##  $ Atributo.1: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Atributo.2: int  0 0 1 0 0 0 0 0 1 0 ...
##  $ Atributo.3: int  0 0 1 0 1 0 0 1 1 1 ...
##  $ Atributo.4: int  1 1 1 1 0 1 0 0 1 1 ...
##  $ Atributo.5: int  1 1 0 0 0 0 0 1 1 1 ...
##  $ Atributo.6: int  1 1 1 1 1 0 1 0 1 1 ...
##  $ Resultado : int  0 0 0 0 0 0 0 0 0 0 ...
```

```
str(arbol3.test)
```

```
## 'data.frame':    10000 obs. of  7 variables:
##  $ Atributo.1: int  0 0 0 0 1 0 0 0 0 0 ...
##  $ Atributo.2: int  0 0 1 0 1 0 0 1 0 0 ...
##  $ Atributo.3: int  1 0 1 0 0 0 0 0 0 0 ...
##  $ Atributo.4: int  1 1 1 0 1 0 1 1 0 1 ...
##  $ Atributo.5: int  0 1 1 1 1 0 1 1 1 1 ...
##  $ Atributo.6: int  1 1 1 1 0 1 0 1 1 1 ...
##  $ Resultado : int  0 0 0 0 1 0 0 0 0 0 ...
```

- **b**. Para comenzar, diseño mi operador predictivo con el conjunto de entrenamiento:

```
fit.arbol3.train <- rpart(formula = Resultado ~ .,
                          data = arbol3.train, method = "class")
```

Chequeo la solución **fit.arbol3.train** sobre el conjunto de entrenamiento:

```
printcp(fit.arbol3.train)
```

```
##
## Classification tree:
## rpart(formula = Resultado ~ ., data = arbol3.train, method = "class")
##
## Variables actually used in tree construction:
## [1] Atributo.1 Atributo.2 Atributo.5 Atributo.6
##
## Root node error: 1498/15000 = 0.099867
##
## n= 15000
##
##         CP nsplit rel error  xerror     xstd
## 1 0.527370      0   1.00000 1.00000 0.024513
## 2 0.205607      1   0.47263 0.47263 0.017338
## 3 0.056742      2   0.26702 0.26702 0.013172
## 4 0.019359      3   0.21028 0.21028 0.011723
## 5 0.010000      4   0.19092 0.19826 0.011390
```

"fit.arbol3.train" puede ser usado para predecir usando los otros dataframes diseñados. Entonces quiero predecir la variable **Resultado** del dataframe **arbol3.test**:

```
prediccion.arbol3.test <- predict(fit.arbol3.train, arbol3.test, type = "class")

# Veo los resultados:
class(prediccion.arbol3.test)
```

```
## [1] "factor"
```

```
str(prediccion.arbol3.test)
```

```
##  Factor w/ 2 levels "0","1": 1 1 1 1 2 1 1 1 1 1 ...
##  - attr(*, "names")= chr [1:10000] "30001" "30002" "30003" "30004" ...
```

```
summary(prediccion.arbol3.test)
```

```
##    0    1
## 9004  996
```

```
length(prediccion.arbol3.test) # OK! 10000 predicciones!
```

```
## [1] 10000
```

Comparo contra los resultados (conocidos) almacenados en la variable **Resultado** del dataframe correspondiente:

```
sum(arbol3.test$Resultado==prediccion.arbol3.test)/length(prediccion.arbol3.test)
```

```
## [1] 0.9806
```

lo cual constituye una muy buena predicción.

Ahora, queremos atacar el problema de podar nuestro árbol de modo de no causar un impacto directo en la tasa de éxito anteriormente obtenida.

Podar un árbol tiene por objetivo evitar el sobreajuste de los datos. Por lo general, se quiere seleccionar el tamaño de árbol que minimice el error de validación cruzada, es decir la columna "xerror" que muestra la función **printcp**:

```
printcp (fit.arbol3.train)
```

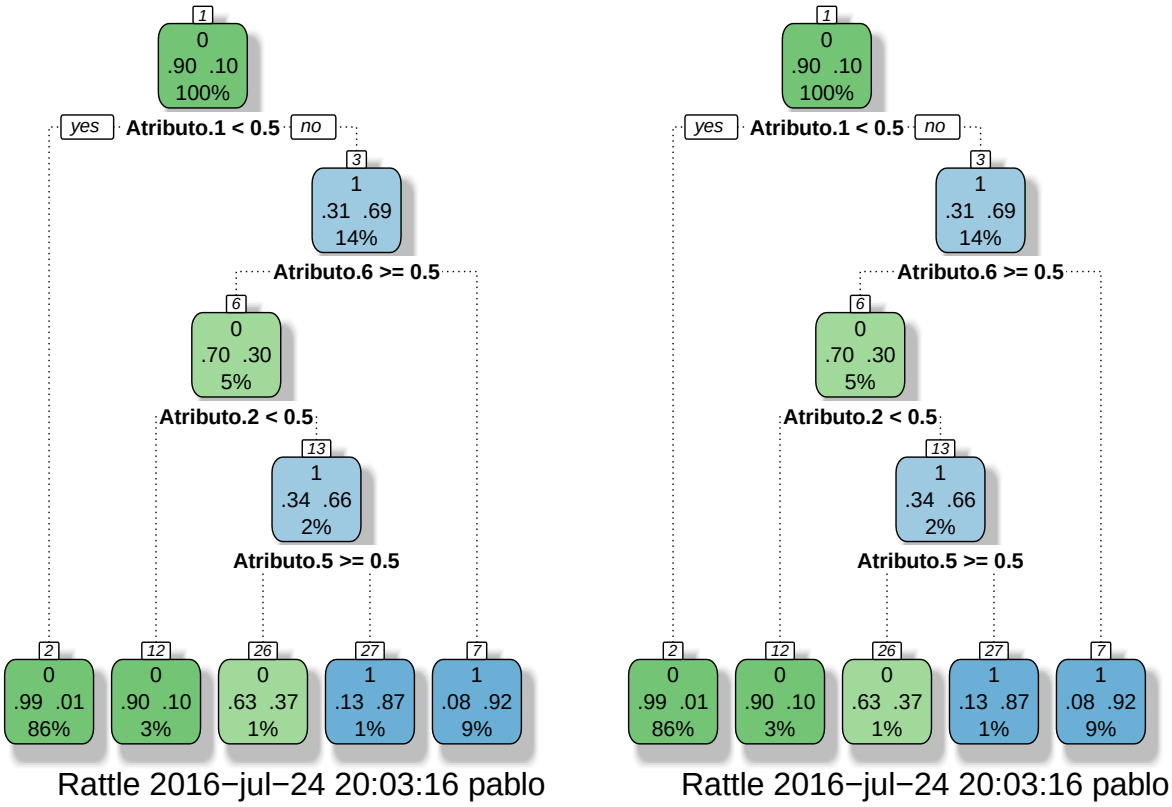
```
##
## Classification tree:
## rpart(formula = Resultado ~ ., data = arbol3.train, method = "class")
##
## Variables actually used in tree construction:
## [1] Atributo.1 Atributo.2 Atributo.5 Atributo.6
##
## Root node error: 1498/15000 = 0.099867
##
## n= 15000
##
##         CP nsplit rel error  xerror     xstd
## 1 0.527370      0   1.00000 1.00000 0.024513
## 2 0.205607      1   0.47263 0.47263 0.017338
## 3 0.056742      2   0.26702 0.26702 0.013172
## 4 0.019359      3   0.21028 0.21028 0.011723
## 5 0.010000      4   0.19092 0.19826 0.011390
```

Examinando los resultados de error con validación cruzada, debemos seleccionar el parámetro de complejidad "cp" asociado con el error mínimo "xerror", y colocarlo en la función **prune()**. En este caso es fácil de identificar, es el "cp" correspondiente al índice número 5. Se muestra también un código alternativo.

```
fit.arbol3.train.prune <- prune(fit.arbol3.train,
                                cp = fit.arbol3.train$cptable
                                [which.min(fit.arbol3.train$cptable[,"xerror"]),
                                "CP"])
```

Hagamos la comparación del árbol sin/con poda incluída:

```
par(mfrow = c(1,2))
fancyRpartPlot(fit.arbol3.train)          # sin poda
fancyRpartPlot(fit.arbol3.train.prune)    # con poda
```




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

Parecen iguales, chequeo con **identical**

```
identical(fit.arbol3.train, fit.arbol3.train.prune)
```

```
## [1] TRUE
```

**rpart** hizo la poda automáticamente al diseñar el árbol inicialmente, en acuerdo con lo ya visto en anteriores ejercicios. Por eso ambas soluciones coinciden.

**Ejercicio árbol 3 repetido, pero usando "arbol3.prune" como conjunto de entrenamiento:**

```
fit.arbol3.prune <- rpart(formula = Resultado ~ .,
                          data = arbol3.prune, method = "class")
```

Chequeo el resultado del diseño del árbol:

```
printcp(fit.arbol3.prune)
```

```
##
## Classification tree:
## rpart(formula = Resultado ~ ., data = arbol3.prune, method = "class")
```

```
##
## Variables actually used in tree construction:
## [1] Atributo.1 Atributo.2 Atributo.5 Atributo.6
##
## Root node error: 1438/15000 = 0.095867
##
## n= 15000
##
##         CP nsplit rel error  xerror     xstd
## 1 0.451321      0   1.00000 1.00000 0.025075
## 2 0.309458      1   0.54868 0.54868 0.019013
## 3 0.028164      2   0.23922 0.23922 0.012749
## 4 0.010000      4   0.18289 0.18289 0.011178
```

Uso este árbol para predecir con el dataframe "arbol3.test"

```
prediccion.arbol3.test.prune <- predict(fit.arbol3.prune, arbol3.test, type = "class")
```

Y calculo la precisión lograda:

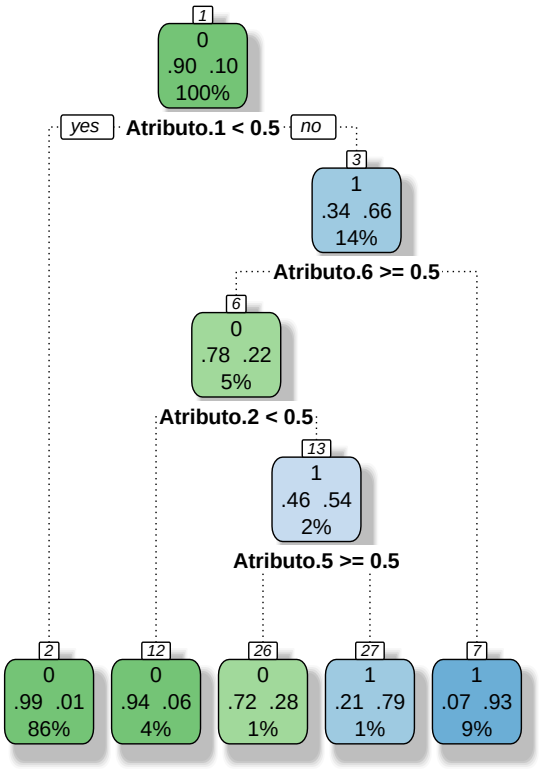
```
 sum(arbol3.test$Resultado==prediccion.arbol3.test.prune)/length(prediccion.arbol3.test.prune)
```

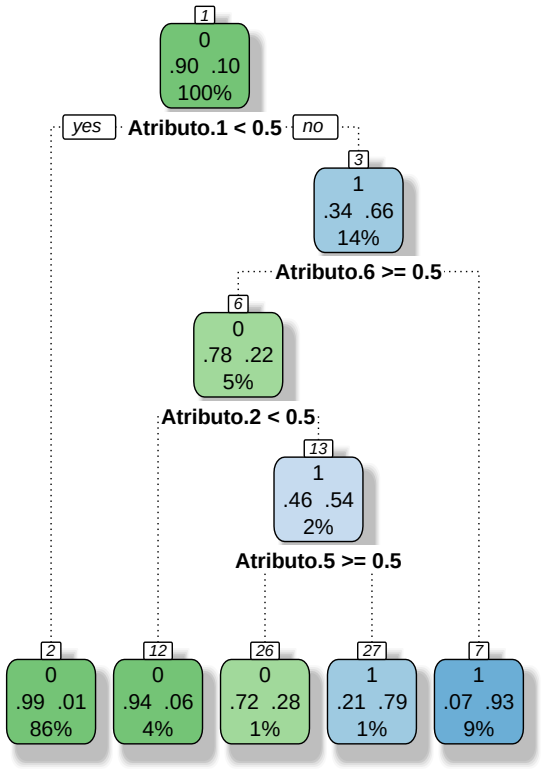
```
## [1] 0.9806
```

Efectúo la poda y chequeo:

```
 fit.arbol3.prune.prune <- prune(fit.arbol3.prune,
                                 cp = fit.arbol3.prune$cptable
                                 [which.min(fit.arbol3.prune$cptable[,"xerror"]),
                                 "CP"])

par(mfrow = c(1,2))
fancyRpartPlot(fit.arbol3.prune) # sin poda
fancyRpartPlot(fit.arbol3.prune.prune) # con poda
```




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

Claramente son distintos. Calculo como cambia la precisión lograda:

```
prediccion.arbol3.test.prune.prune <- predict(fit.arbol3.prune.prune,
                                              arbol3.test, type = "class")
sum(arbol3.test$Resultado==prediccion.arbol3.test.prune.prune)/length(prediccion.arbol3.test.prune.prune
```

```
## [1] 0.9806
```

---