Metodos basados en arboles: CART y Random Forest

Jaqueline Girabel

En este trabajo utilizaremos los métodos CART y RF, que hacia el final serán comparados con el de regresión logística, para modelar un clasificador sobre el conjunto de datos **vidrios**, cuya variable respuesta indica diferenes tipos de vidrios dependientes de una serie de características. El estudio de tipo de vidrios fue motivado por investigación criminológica, dado que, cuando una clase de vidrio se identifica correctamente, puede ser utilizada como evidencia.

La variable respuesta, *Glass*, puede incluir hasta 7 tipos de vidrios distintos. Sin embargo, en este conjunto de datos, de las 214 observaciones ninguna corresponde a la clase de Tipo 4. Damos a continuación el detalle de cada clase en la variable *Glass*.

Tipo 1: building_windows_float_processed
Tipo 2: building_windows_non_float_processed
Tipo 3: vehicle_windows_float_processed
Tipo 4: vehicle_windows_non_float_processed Tipo 5: containers
Tipo 6: tableware
Tipo 7: headlamps.

```
dim(vidrios)
```

```
## [1] 214  10
```

```
head(vidrios)
```

```
##        RI    Na   Mg   Al    Si    K   Ca Ba   Fe  Glass
## 1 1.52101 13.64 4.49 1.10 71.78 0.06 8.75  0 0.00 Tipo 1
## 2 1.51761 13.89 3.60 1.36 72.73 0.48 7.83  0 0.00 Tipo 1
## 3 1.51618 13.53 3.55 1.54 72.99 0.39 7.78  0 0.00 Tipo 1
## 4 1.51766 13.21 3.69 1.29 72.61 0.57 8.22  0 0.00 Tipo 1
## 5 1.51742 13.27 3.62 1.24 73.08 0.55 8.07  0 0.00 Tipo 1
## 6 1.51596 12.79 3.61 1.62 72.97 0.64 8.07  0 0.26 Tipo 1
```

Notamos que no hay valores ausentes, contabilizándolos por columna:

```
library(purrr)
NAs_por_columna<-map_dbl(vidrios, function(x){sum(is.na(x))})
NAs_por_columna #No  hay valores ausentes
```

```
##    RI    Na    Mg    Al    Si     K    Ca    Ba    Fe Glass
##     0     0     0     0     0     0     0     0     0     0
```

En este conjunto de datos, las clases de la variable respuesta no están representadas de manera proporcionada. En particular, no hay datos de la clase de Tipo 4 y la clase de Tipo 6 se corresponde sólo con 9 observaciones, que serán removidas del dataset porque sólo aportarán ruido a la hora de modelar un clasificador.

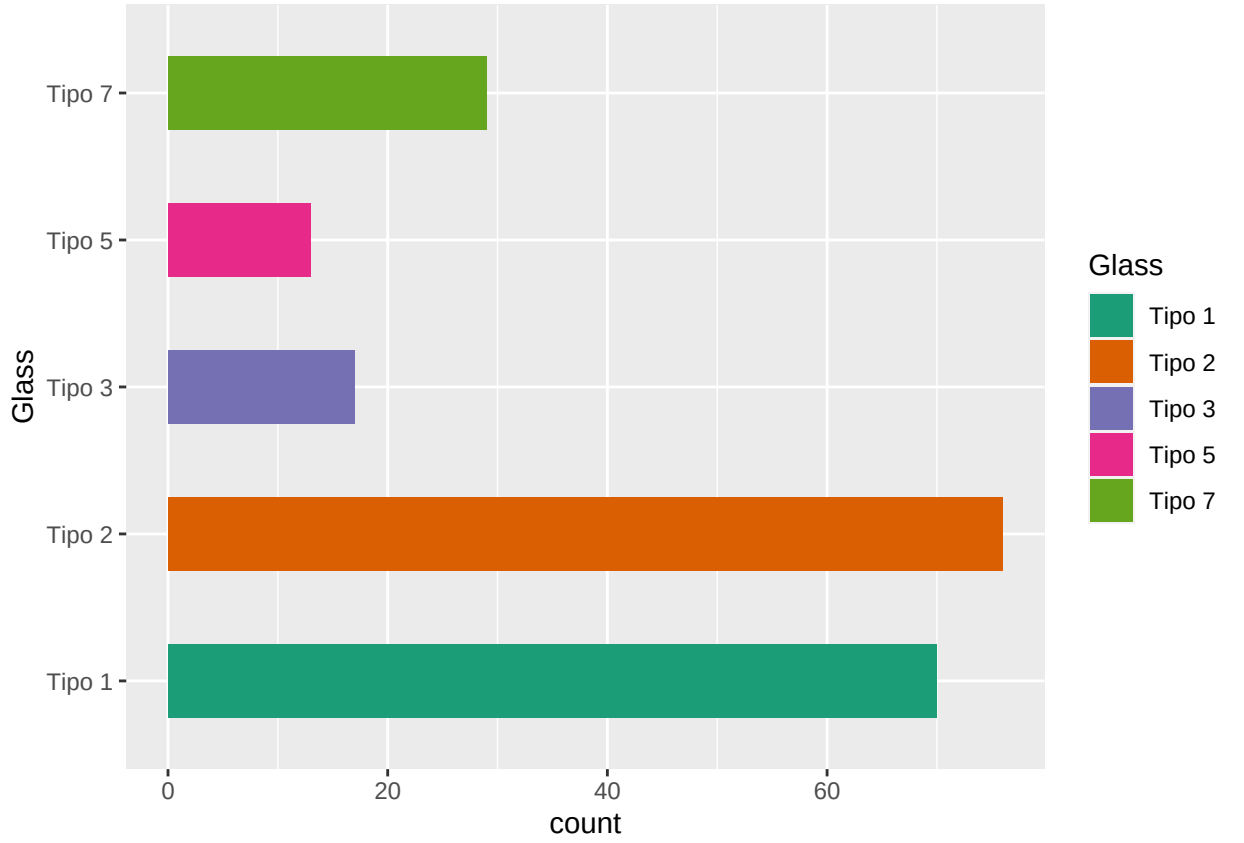
```
table(vidrios$Glass)
```

```
##
## Tipo 1 Tipo 2 Tipo 3 Tipo 5 Tipo 6 Tipo 7
##     70     76     17     13      9     29
```

```
vidrios<-vidrios[Glass!="Tipo 6",]
vidrios$Glass<-factor(vidrios$Glass)
```

En el siguiente gráfico de barras puede verse que las clases de la variable Glass están completamente desbalanceadas, siendo las clases de Tipo 1 y 2 las más frecuentes.

```
library(ggplot2)

ggplot(vidrios, aes(x=Glass))+geom_bar(aes(fill=Glass),
width=0.5)+coord_flip()+scale_fill_brewer(palette="Dark2")
```



Para tratar este problema, construimos un nuevo set de datos **vidrios_balanceado** mediante la técnica de oversampling, y será el que utilizaremos al momento de modelar un clasificador. Lo que obtenemos es un conjunto de datos en el que todas las clases están igualmente representadas, todas con igual frecuencia que aquella cuya frecuencia es máxima en el set de datos original. De esta manera, se conservan los datos originales intactos, y los datos que se incorporan son generados aleatoriamente resampleando en las observaciones de cada clase.

```
library(UBL)

set.seed(123)
vidrios_balanceado <- RandOverClassif(Glass ~ ., vidrios)
```

```
table(vidrios_balanceado$Glass)
```

```
##
## Tipo 1 Tipo 2 Tipo 3 Tipo 5 Tipo 7
##     76     76     76     76     76
```

Vamos a emplear tres modelos para construir un clasificador de tipo de vidrios para este set de datos. El primero será ajustado con el método CART para construir un árbol de decisión al que se le aplicará un criterio de poda para reducir su complejidad, de ser necesario. Para el segundo modelo usaremos Random Forest, generalmente caracterizado por ser un modelo más independiente de los datos. Por último, aplicaremos una regresión logística, que en este caso deberá ser una regresión multinomial.

Los compararemos mediante el error obtenido por validación cruzada de tipo *K-folds*, con K=10.

Comencemos con el **modelo.cart**.

```
library("tree")

modelo.cart<-tree(Glass ~., vidrios_balanceado)
summary(modelo.cart)
```

```
##
## Classification tree:
## tree(formula = Glass ~ ., data = vidrios_balanceado)
## Variables actually used in tree construction:
## [1] "Mg" "Na" "Al" "Si" "Ca" "RI"
## Number of terminal nodes:  16
## Residual mean deviance:  0.6388 = 232.5 / 364
## Misclassification error rate: 0.1211 = 46 / 380
```

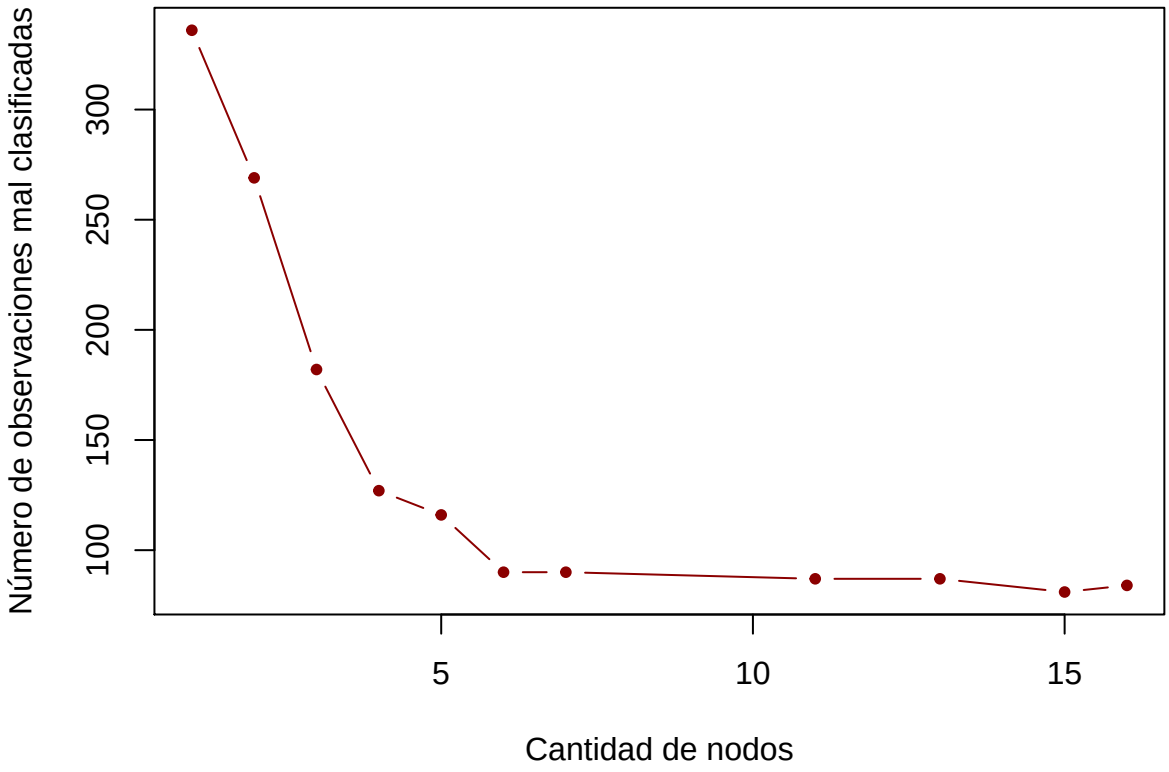
En el resumen anterior se pueden ver las variables de decisión utilizadas para construir el árbol, la cantidad de regiones (nodos) en las que fue subdividido el conjunto de datos, y el error de clasificación, siendo este último 0.1221.

Para asegurarnos que nuestro modelo no sufra de varianza alta, nos quedaremos con el resultado de aplicar un criterio de poda en el árbol obtenido. Lo que hacemos es evaluar el error en función de la cantidad de nodos - y de k, que es el parámetro que define la complejidad de la poda -, mediante cross validation.

```
# prune.misclass para podar el ?rbol obtenido previamente
set.seed(123)
cv.cart<-cv.tree(modelo.cart, FUN = prune.misclass)
```

En el gráfico que sigue se ilustra la evolución del error:

```
plot(cv.cart$size, cv.cart$dev, type = "b", col="darkred",
     ylab="Número de observaciones mal clasificadas", xlab="Cantidad de nodos", pch=20)
```



La mínima cantidad de observaciones mal clasificadas es

```
min <- which.min(cv.cart$dev) #indice del minimo error
cv.cart$dev[min]  #cantidad de errores
```

```
## [1] 81
```

y la cantidad de nodos asociada es

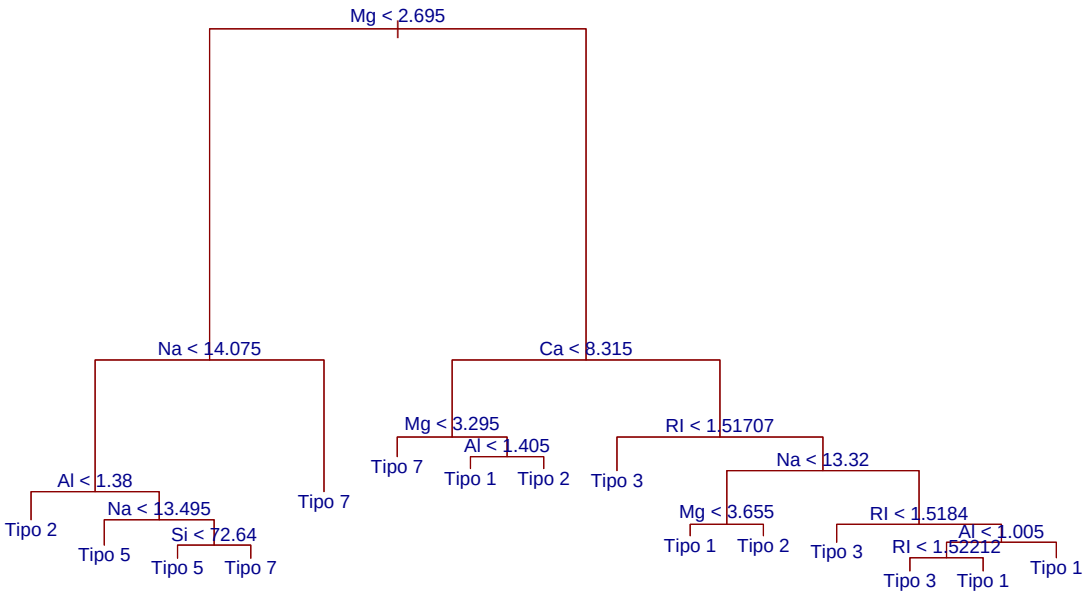
```
## [1] 15
```

Luego, el error obtenido por validación cruzada utilizando el modelo CART podado con 15 nodos es

```
## [1] 0.2131579
```

Notamos que este es un árbol con un nodo menos que el original, y el error práticamente se duplicó. Vemos en el siguiente gráfico el modelo de 15 nodos, que será referido como **modelo.cart.podado**.

```
modelo.cart.podado<-prune.misclass(modelo.cart,best =cv.cart$size[min])
plot(modelo.cart.podado, col="darkred", pch=20)
text(modelo.cart.podado, pretty =0, col="darkblue")
```



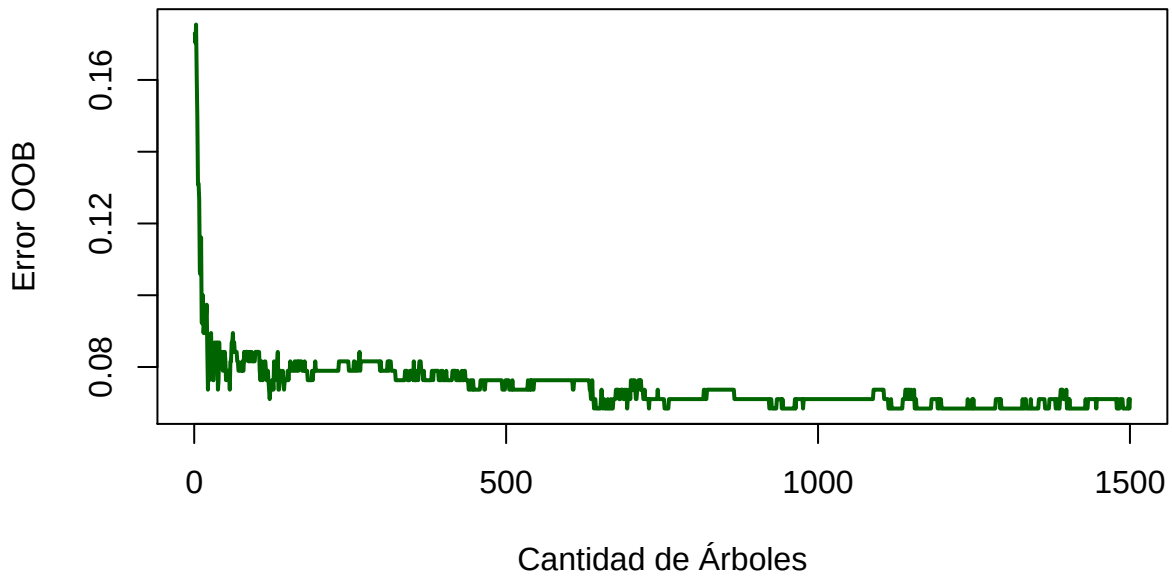
Con el modelo.cart.podado tenemos la siguiente matriz de confusión en el set de datos original **vidrios**.

```
predicciones<-predict(modelo.cart.podado, vidrios, type="class")
table(vidrios$Glass, predicciones)
```

```
##          predicciones
##           Tipo 1 Tipo 2 Tipo 3 Tipo 5 Tipo 7
##   Tipo 1     55      6      9      0      0
##   Tipo 2     16     48      5      5      2
##   Tipo 3      0      0     17      0      0
##   Tipo 5      0      0      0     13      0
##   Tipo 7      1      1      0      0     27
```

Veamos ahora qué ocurre con el método Random Forest. Lo primero que hacemos es identificar el número de árboles (réplicas bootstrap) a utilizar.

```
library(randomForest)

set.seed(123)
modelo.rf<-randomForest(Glass~., vidrios_balanceado, ntree=1500 )

plot(modelo.rf$err.rate[,1], type="l",
     col="darkgreen", ylab = "Error OOB",
     xlab = "Cantidad de Árboles", lwd=2)
```



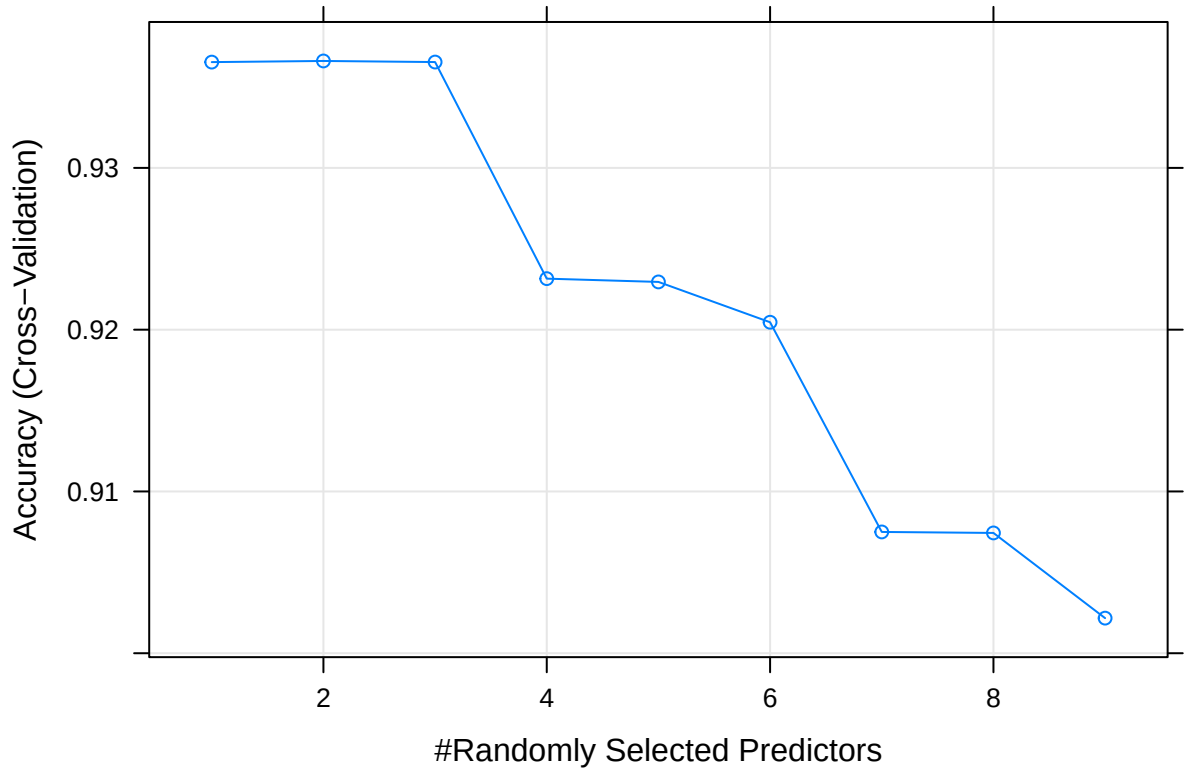
Definimos ntree=600 y usamos la funcionalidad *train* para ajustar el mejor modelo evaluado por validación cruzada, en función de la cantidad de variables explicativas que utiliza el método para construir un árbol.

```
library(caret)

hiperparametro<-expand.grid(.mtry=seq(from=1, to=ncol(vidrios)-1, by=1))

ajuste<-trainControl(method='cv', number = 10)

set.seed(123)
modelo.rf <- train(Glass~., vidrios_balanceado,
                   method = 'rf',
                   tuneGrid = hiperparametro,
                   trControl = ajuste,
                   ntree=600)

modelo.rf
```

```
## Random Forest
##
## 380 samples
##   9 predictor
##   5 classes: 'Tipo 1', 'Tipo 2', 'Tipo 3', 'Tipo 5', 'Tipo 7'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 341, 341, 343, 341, 342, 345, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   1     0.9365383  0.9206036
##   2     0.9366094  0.9206529
##   3     0.9365419  0.9205759
##   4     0.9231548  0.9038410
##   5     0.9229521  0.9035870
##   6     0.9204591  0.9004594
##   7     0.9074966  0.8842540
##   8     0.9074362  0.8841594
##   9     0.9021694  0.8775904
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

El mejor modelo en términos de la exactitud se obtiene considerando dos variables predictoras, seleccionadas aleatoriamente en cada split.

```r
plot(modelo.rf)
```



Luego, el error de predicci?n es

```r
round(1-modelo.rf[["results"]][2,2],4)
```

```
## [1] 0.0634
```

Por último, comparemos los resultados anteriores con los obtenidos por un clasificador modelado con regresión logística multinomial.

```
# sin penalizaci?n
decay<-expand.grid(decay=0)

set.seed(123)
modelo.multinom<- train(Glass~., vidrios_balanceado,
                   method = 'multinom',
                   trControl = ajuste,
                   tuneGrid=decay,
                   trace = FALSE)
modelo.multinom
```

```
## Penalized Multinomial Regression
##
## 380 samples
##   9 predictor
##   5 classes: 'Tipo 1', 'Tipo 2', 'Tipo 3', 'Tipo 5', 'Tipo 7'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 341, 341, 343, 341, 342, 345, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.6964349  0.6207304
##
## Tuning parameter 'decay' was held constant at a value of 0
```

El error de predicción del **modelo.multinom** es

```
1-modelo.multinom[["results"]][1,2]
```

```
## [1] 0.3035651
```

Podemos concluir que, en términos del error obtenido por validación cruzada, el mejor modelo es **modelo.rf**, seguido del **modelo.cart.podado**.