Informe Becas

Miguel Figueira

12 de marzo de 2016

```
## Loading required package: class

## Loading required package: dplyr

## Warning: package 'dplyr' was built under R version 3.2.3

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Loading required package: rpart

## Loading required package: rpart.plot

## Warning: package 'rpart.plot' was built under R version 3.2.3

## Loading required package: RWeka

## Warning: package 'RWeka' was built under R version 3.2.3

## Loading required package: caret

## Warning: package 'caret' was built under R version 3.2.3

## Loading required package: lattice

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 3.2.3
```

Preprocesamiento

1. Eliminar features que no son necesarios:

```
dat$jReprobadas <- NULL
dat$dHabitacion <- NULL
dat$sugerencias <- NULL
dat$aEconomica <- NULL
dat$cDireccion <- NULL
dat$oSolicitudes <- NULL
```

```
dat$pReside <- NULL
dat$cIdentidad <- NULL
```

2. Acomodar la columna grOdontologicos que tiene una instancia con "o" en vez de 0(cero)

```
dat$grOdontologicos[dat$grOdontologicos == "o"] <- as.character(0)
dat$grOdontologicos <- as.integer(dat$grOdontologicos)
```

3. Comparar la fecha con una fecha actual para obtener la edad , además de eliminar las instancias que tienen una edad menor a 5 (son 2 instancias) las cuales no tienen sentido y pueden "dañar" el modelo

```
actual <- as.Date("01/03/2016",format("%d/%m/%Y"))
a <- rep(as.Date("14/03/1979",format="%d/%m/%Y"),length(dat$fNacimiento))
edad <- rep(NA,length(dat$fNacimiento))
for(i in 1:length(dat$fNacimiento)){
  a[i] <- as.Date(dat$fNacimiento[i],format="%d/%m/%Y")
  edad[i] <- as.integer(actual-a[i]) %/% 365
}
dat$fNacimiento <- NULL
dat$edad <- edad

dat <- dat[dat$edad > 5, ]
```

4. Eliminar la instancia que tiene el "mIngreso" = 1 ya que es la única instancia que tiene este modo de ingreso por lo tanto los modelos no pueden "aprender"y en caso de que sea parte del testing y si es parte del training no es lo suficientemente significativo.

```
dat <- dat[dat$mIngreso != 1, ]
```

5. Crear la data de entrenamiento y de prueba , manteniendo el mismo porcentaje de mIngreso que en el dataset original:

```
dat <- dat[dat$mIngreso != 1, ]



dat0 <- dat[dat$mIngreso == 0, ]
dat2 <- dat[dat$mIngreso == 2, ]
dat3 <- dat[dat$mIngreso == 3, ]

sampl0 <-  sample(nrow(dat0), floor(nrow(dat0) * 0.75))
sampl2 <-  sample(nrow(dat2), floor(nrow(dat2) * 0.75))
sampl3 <-  sample(nrow(dat3), floor(nrow(dat3) * 0.75))

dat_train0 <- dat0[sampl0, ]
dat_train2 <- dat2[sampl2, ]
dat_train3 <- dat3[sampl3, ]
```

```
dat_train <- rbind(dat_train0,dat_train2,dat_train3)

dat_test0 <- dat0[-sampl0, ]
dat_test2 <- dat2[-sampl2, ]
dat_test3 <- dat3[-sampl3, ]

dat_test <- rbind(dat_test0,dat_test2,dat_test3)
```

6. Convertir mIngreso de caracter a factor:

```
dat_train$mIngreso <- as.factor(dat_train$mIngreso)
dat_test$mIngreso <- as.factor(dat_test$mIngreso)
```

KNN

1. Normalizar los datos menos la columna label que en este caso es la columna "mIngreso":

```
c <- 1:length(dat)
c <- c[-5]

knn_train <- as.data.frame(lapply(dat_train[c], normalize))
knn_test <- as.data.frame(lapply(dat_test[c], normalize))
#normalizar todas las columnas menos el feature label


knn_train_labels <- dat_train[ ,5]
knn_test_labels <- dat_test[ ,5]
```

2. Haremos KNN con los valores más recomendados 8,9,10 y evaluamos su precisión

Ejemplo de como se calcula el KNN , matriz de confusion y la precisión (igual con k=9 y k=10)

```
prc_test_pred <- knn(train = knn_train, test = knn_test,cl =
knn_train_labels, k=8)


confusionMatrix.knn8<- table(knn_test_labels,prc_test_pred)

accuracy.knn8 <- (confusionMatrix.knn8[1,1] + confusionMatrix.knn8[2,2] +
                      confusionMatrix.knn8[3,3])/
sum(confusionMatrix.knn8)
```

Matrices de confusión y precisión de los KNN:

1. K= 8

```
#Matriz de Confusion con K = 8
confusionMatrix.knn8
```

```
##                prc_test_pred
## knn_test_labels  0  2  3
##               0  8  0 10
##               2  0  0  2
##               3  8  0 20
```

```
#Precision con K = 8
accuracy.knn8
```

```
## [1] 0.5833333
```

```
comparar[1] <- accuracy.knn8
```

2. K = 9

```
#Matriz de Confusion con K = 9
confusionMatrix.knn9
```

```
##                prc_test_pred
## knn_test_labels  0  2  3
##               0  4  0 14
##               2  0  0  2
##               3  9  0 19
```

```
#Precision con K = 9
accuracy.knn9
```

```
## [1] 0.4791667
```

```
comparar[2] <- accuracy.knn9
```

3. K = 10

```
#Matriz de Confusion con K = 10
confusionMatrix.knn10
```

```
##                prc_test_pred
## knn_test_labels  0  2  3
##               0  7  0 11
##               2  0  0  2
##               3 10  0 18
```

```
#Precision con K = 10
accuracy.knn10
```

```
## [1] 0.5208333
```

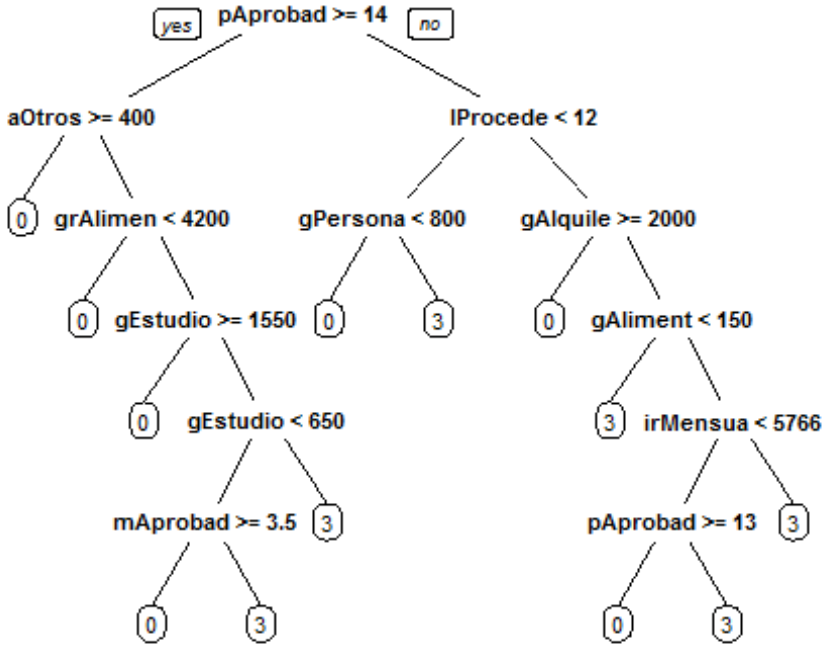
```
comparar[3] <- accuracy.knn10
```

Arboles de decision

Probaremos con 3 arboles, en uno variando el minsplit y en otro el cp

1. minsplit = 10 , cp 0.001

```
#Arbol con minsplit = 10, cp = 0.001
rpart.plot(tree)
```




```
#Matriz de Confusion, Arbol con minsplit = 10, cp = 0.001
confusionMatrix.tree
```

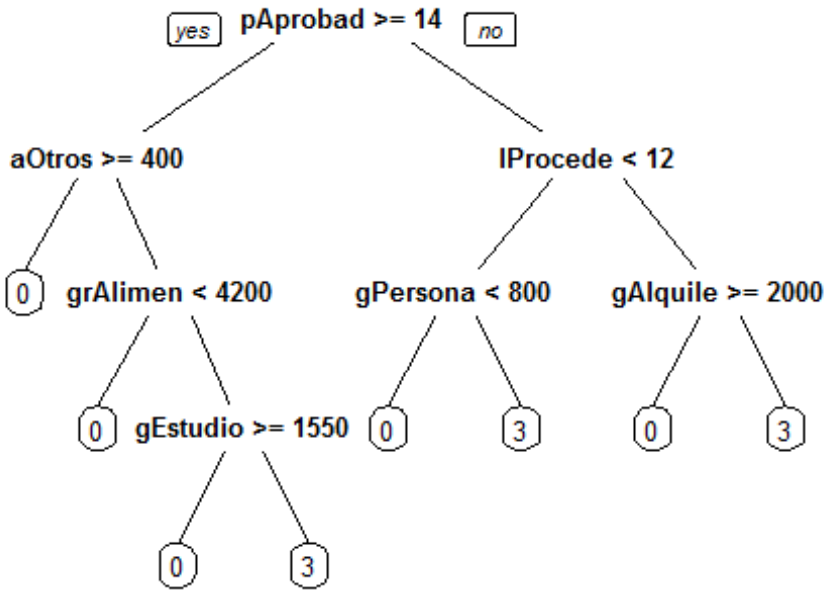
```
##
##      0  2  3
##   0 11  0  7
##   2  2  0  0
##   3  8  0 20
```

```
#Precision, Arbol con minsplit = 10, cp = 0.001
accuracy.tree
```

```
## [1] 0.6458333
```

2. minsplit = 10 , cp 0.001

```
#Arbol
rpart.plot(tree2)
```




```
#Matriz de Confusion, Arbol con minsplit = 15, cp = 0.001
confusionMatrix.tree2
```

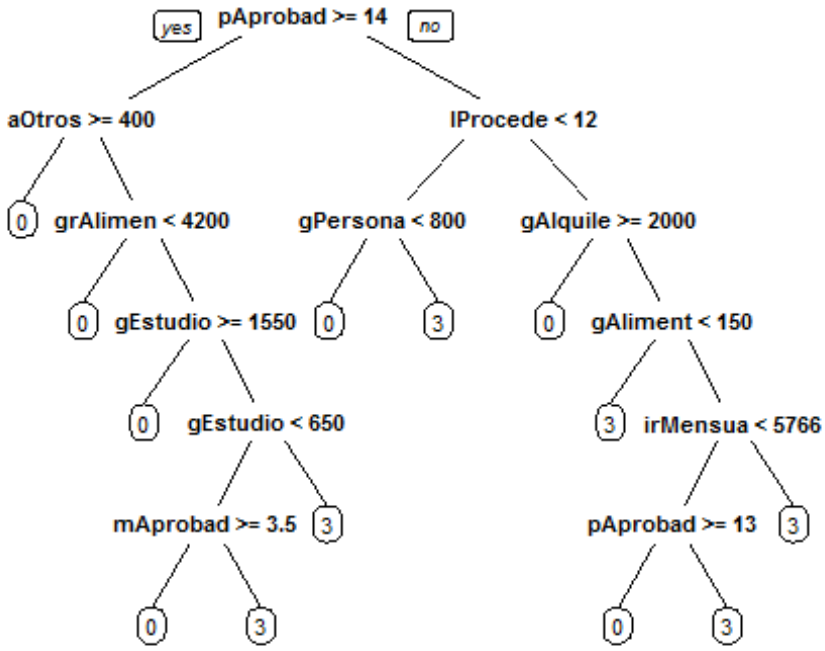
```
##
##      0  2  3
##   0  7  0 11
##   2  2  0  0
##   3  7  0 21
```

```
#Precision, Arbol con minsplit = 15, cp = 0.001
accuracy.tree2
```

```
## [1] 0.5833333
```

3. minsplit = 10 , cp = 0.01

```
#Arbol con minsplit = 10, cp = 0.01
rpart.plot(tree3)
```




```
#Matriz de Confusion, Arbol con minsplit = 15, cp = 0.01
confusionMatrix.tree3
```

```
##
##      0  2  3
##   0 11  0  7
##   2  2  0  0
##   3  8  0 20
```

```
#Precision, Arbol con minsplit = 15, cp = 0.01
accuracy.tree3
```

```
## [1] 0.6458333
```

Clasification Rules

1.JRip: JRip implements a propositional rule learner, "Repeated Incremental Pruning to Produce Error Reduction" (RIPPER), as proposed by Cohen (1995)

```
#confusion matrix
confusionMatrix.JRip
```

```
##
##      0  2  3
##   0  8  0 10
##   2  0  0  2
##   3  0  0 28
```

```
#Precision
accuracy.JRip
```

```
## [1] 0.75
```

```
comparar[7] <- accuracy.JRip
```

2. OneR builds a simple 1-R classifier, see Holte (1993)

```
#Matriz de Confusion OneR
confusionMatrix.OneR
```

```
##
##      0  2  3
##   0 11  0  7
##   2  0  0  2
##   3 13  0 15
```

```
#Precision OneR
accuracy.OneR
```

```
## [1] 0.5416667
```

3. PART generates PART decision lists using the approach of Frank and Witten (1998).

```
#confusion matrix PART
confusionMatrix.PART
```

```
##
##      0  2  3
##   0  6  1 11
##   2  2  0  0
##   3 11  2 15
```

```
#Precision PART
accuracy.PART
```

```
## [1] 0.4375
```

Es importante destacar que solo tomo en cuenta porque en este caso no es de relevancia la sensitividad o la especificidad como sí podría serlo en casos como detectar o no el cancer de una persona

```
#el mejor valor es
comparar <- comparar == max(comparar)
comparar
```

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

```
#donde de TRUE ES EL MEJOR O MEJORES VALORES
# de izq a derecha es
```