becas

Andrea Tellería

March 11, 2016

1. Objetivos:

La Organización de Becas Crema mediante la vista minable que fue anteriormente provista, desea realizar un análisis sobre sus datos con la intención de encontrar patrones en ellos que le permitan realizar predicciones sobre nuevos individuos. Con esta idea en mente se hace obvio la necesidad de primero, saber qué se desea buscar, qué se requiere clasificar.

Como lo que se deseada es saber el método de ingreso de los estudiantes, es decir, su clase, entonces el problema se convierte en la búsqueda de un modelo que nos ayude a predecir mediante clasificación.

Para ello se procedieron a estudiar un conjunto de métodos diferentes para conseguir modelos a fin de elegir el mejor de ellos.

2. Solución

2.1 Instalación de librerías necesarias

Primero que nada es necesario la instalación de ciertos paquetes que fueron usados a lo largo del análisis, para ello se procede a verificar si estos ya han sido instalados de no estarlo, realiza la instalación pertinente.

```
######################################################################
# Instalación de Paquetes necesarios
######################################################################

#Creamos la función que recibe los paquetes
install = function(pkg){
  #Si ya está instalado, no lo instala.
  if (!require(pkg, character.only = TRUE)) {
    install.packages(pkg)
    if (!require(pkg, character.only = TRUE)) stop(paste("load failure:", pkg))
  }
}
install("foreach")
```

```
## Loading required package: foreach
```

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

```
#Seleccionamos los archivos que queremos instalar
archive = c("rJava", "shiny", "rmarkdown", "caret", "rpart", "tree", "RWeka", "rpart.plot", "class", "sa
foreach(i = archive) %do% install(i)
```

```
## Loading required package: rJava
```

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

## Loading required package: shiny

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

## Loading required package: rmarkdown

## 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

## Loading required package: rpart

## Loading required package: tree

## Warning: package 'tree' 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: rpart.plot

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

## Loading required package: class

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

## Loading required package: sampling

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

##
## Attaching package: 'sampling'

## The following object is masked from 'package:caret':
##
##     cluster
```

2.2 Cargando la vista minable

Una vez se tienen los paquetes que son necesarios, pasamos a iniciar como tal el análisis sobre nuestra data. Así pues, desde la vista minable creada con anterioridad se extraen los datos para poder analizar los mismos.

```
#####################################################################
# Lectura de archivos y carga de datos
#####################################################################

datos = read.table("data.csv", header = TRUE, sep = ",", dec = ".")
```

2.3 Escogiendo variables

Con los datos ya cargados pasamos a algo de preprocesamiento. En este punto lo más importante es escoger cuáles variables nos van a ayudar a la hora de realizar el modelo y aquellas que no aportan información valiosa. Así pues, se decidió eliminar del conjunto de datos aquellos que fuesen sugerencias hechas por los usuarios de la beca, así como aquellas columnas cuyo tipo de dato no fuese de tipo numérico o factor.

Una de las razones principales para hacer esto, se debe también al tipo de funciones que van a usarse a la hora de buscar el modelo y que son incompatibles como datos de tipo carácter o cadenas de caracteres (*strings*).

Además de esto tenemos que hacer algo con respecto a la variable con la que vamos a formular los modelos, ya que esta clase (mIngreso) se encuentra muy desbalanceada, es particular la clase 1 y 2. Dónde:

1. Clase 0 (Asignado OPSU): 71 Individuos
2. Clase 1 (Convenios Interinstitucionales, nacionales e internacionales): 1 Individuo
3. Clase 2 (Convenios Internos: deportistas, artistas, hijos empleados docente, obreros y Samuel Robinson): 8 Individuos.
4. Clase 3 (Prueba Interna y/o propedéutico): 110 Individuos.

Como puede verse pertenecer a la clase 1 es anómalo, por lo cual se optó por eliminar el registro y tratarlo como un *outlayer*, con lo cuál sólo quedaría tratar con la clase 2.

Para tratar con ella (clase 2), primero se trató de generar más individuos a base de los existentes con el fin de que existiese representación a la hora de crear el *train set* y el respectivo *test set* (conjunto de entrenamiento y prueba respectivamente).

```
#####################################################################
# Preprocesamiento: Escogiendo variables
#####################################################################

#Variables que necesesitan preprocesamiento
datos$pReside <- as.character(datos$pReside)
datos$pReside[datos$pReside == "Esposo (a) Hijos (as) "] <- 8
datos$pReside <- as.factor(datos$pReside)

#Eliminación de columnas
datos$jReprobadas <- NULL
datos$dHabitacion <- NULL
datos$cDireccion <- NULL
datos$oSolicitudes <- NULL
datos$aEconomica <- NULL
datos$sugerencias <- NULL
datos$cIdentidad <- NULL
datos$fNacimiento <- NULL

#Datos anómalos (una clase con un sólo individuo en mIngreso)
datos <- datos[-c(1), ]
```

```
# Replicamos aquellas clases que nos interesan y que se encuentran muy mal representadas.
repvec <- ifelse(datos$mIngreso=='2',3,1)
data.expanded <- datos[rep(row.names(datos), repvec),]
```

2.4 *Set* de entrenamiento y de prueba

Una vez tenemos las variables con las que hemos decidido quedarnos para realizar los modelos, pasamos rápidamente al paso siguiente, determinar el *set* de entrenamiento y el de prueba que se usara para realizar el modelo. Al hacer esta separación, cabe destacar que el problema percibido antes sobre la mala representación se intensificó.

Sabemos que la variable que deseamos predecir es sobre el método de ingreso de los estudiantes que se benefician con el servicio de becas (variable mIngreso). Al mismo tiempo, la mayoría de estos estudiantes vienen de dos de las clases (0 y 3), como se explicó anteriormente. A pesar de todas las medidas tomadas con anterioridad, la mala representación de la clase 2 sigue existiendo, por esto se tomó la decisión de tratar con un particionamiento estratificado.

Para hacer esto primero se creó un vector de pesos, el cual pasaría a indicar la probabilidad de que un tipo se clase fuese escogido de entre la data original. De esta forma se aseguró que la clase 2 estuviese representada dentro del modelo predictivo que desease usarse.

```
#####################################################################
# Separamos la data en un trainig set y su respectivo test set
#####################################################################

#Matriz de pesos
p<-(1/(length(data.expanded$mIngreso)-8+(8*2)))
weight <- rep(0, length(data.expanded$mIngreso))
weight <- ifelse(data.expanded$mIngreso=='2',p*2,p)

#Tamaño de la muestra
t_size <- floor(0.7 * length(data.expanded$mIngreso))

#Patición
t_indx <- sample(seq_len(length(data.expanded$mIngreso)), size = t_size, prob = weight)
train <- data.expanded[t_indx, ]
test <- data.expanded[-t_indx,]
```

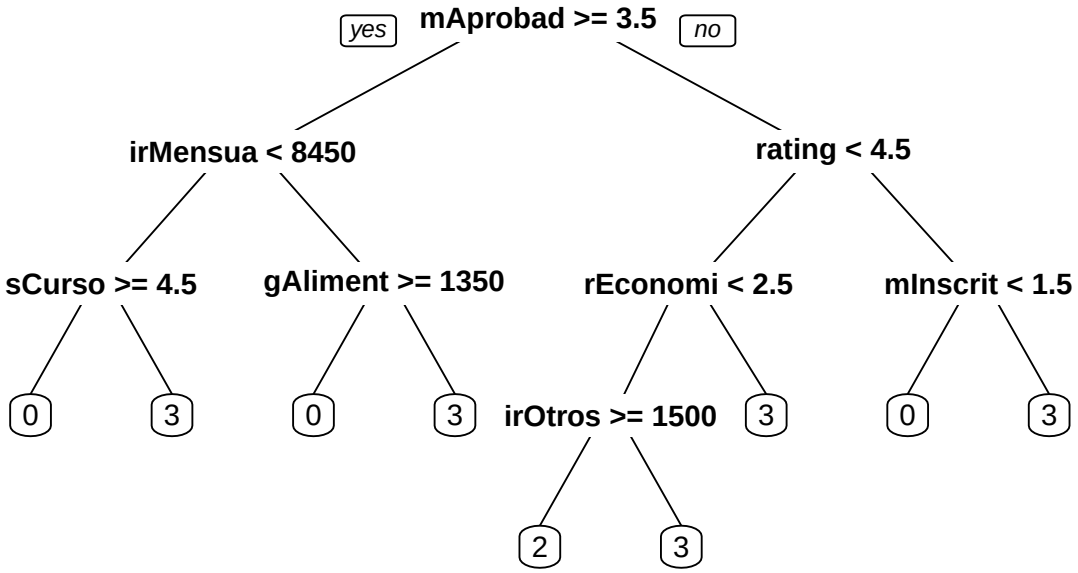
Por medio de la función *sample* y con ayuda de la matriz de pesos se consiguió elegir una muestra en la que la clase dos estuviese, al menos, representada. Para el tamaño del set de entrenamiento se decidió usar el 75% de la data original, dónde el 25% restante se convirtió en el set de prueba que se usará para probar los modelos a usarse.

2.5 Árbol de decisión

Para realizar el árbol de decisión se usó del paquete provisto para R: **rparts**. Específicamente, se hizo uso de su función *rpart*. Se aclaró el método clase y se pasó a crear de un árbol de clasificación. Una vez hecho el modelo, con ayuda del paquete **rpart.plot** se hizo una visualización del árbol.

```
#####################################################################
# Buscando un árbol de decisión con rpart
#####################################################################
```

```
tree <- rpart(train$mIngreso ~ . , data = train, method = 'class')
rpart.plot(tree)
```



2.5.1: Matriz de confusión

Una vez tenemos el modelo pasa a ser necesario poner este prueba, para esto tenemos nuestro *set* de prueba y con ayuda de la función *predict*, se pasó a realizar la predicción de las clases la cual depende de nuestro árbol, previamente creado.

Para crear la matriz de confusión se usó del paquete **caret**, el cual ofrecía la función *confusionMatrix*, junto con el modelo creado y de la información del método de ingreso del conjunto de prueba, se realizó la matriz. Cabe destacar que antes de hacer uso de estas variables se tuvo cuidado de pasar ambas a un tipo de dato factor para que fuesen reconocidas por la función.

Una vez realizada la matriz se imprime la información contenida en ésta.

```
#####################################################################
# Se predice y se busca la matriz de confusión
#####################################################################

pred <- predict(tree, type="class")
model.tree <- pred[1:length(test$mIngreso)]
mIngreso <- as.factor(test$mIngreso)
mcon.tree <- confusionMatrix(model.tree, mIngreso)
mcon.tree
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  2  3
##          0  6  0 10
##          2  1  0  3
##          3 16  4 22
##
## Overall Statistics
##
##                Accuracy : 0.4516
##                  95% CI : (0.3248, 0.5832)
##     No Information Rate : 0.5645
##     P-Value [Acc > NIR] : 0.9721
##
##                   Kappa : -0.0593
##  Mcnemar's Test P-Value : 0.4703
##
## Statistics by Class:
##
##                      Class: 0 Class: 2 Class: 3
## Sensitivity           0.26087  0.00000   0.6286
## Specificity           0.74359  0.93103   0.2593
## Pos Pred Value        0.37500  0.00000   0.5238
## Neg Pred Value        0.63043  0.93103   0.3500
## Prevalence            0.37097  0.06452   0.5645
## Detection Rate        0.09677  0.00000   0.3548
## Detection Prevalence  0.25806  0.06452   0.6774
## Balanced Accuracy     0.50223  0.46552   0.4439
```

2.6 K Vecinos más cercanos

Una vez más con nuestro conjunto de entrenamiento y de prueba pasamos a realizar un nuevo modelo de clasificación, esta vez el método no es otro que K vecinos más cercanos, para usar éste se decidió por la función *knn* provista por el paquete **class**.

Debido a ciertas restricciones del paquete es necesario asegurarse de los tipos de datos que se manejan sean compatibles con la función *knn*. Una vez se realiza esto se procede a realizar el modelo como tal.

```
#####################################################################
# Procesamos algunos tipos de datos para poder usar KNN
#####################################################################

train$grOdontologicos <- as.integer(train$grOdontologicos)
train$pReside <- as.integer(train$grOdontologicos)

test$grOdontologicos <- as.integer(test$grOdontologicos)
test$pReside <- as.integer(test$grOdontologicos)
```

2.6.1 Creando el modelo con *knn*

Una vez estamos seguros de que nuestros tipos de datos son compatibles pasamos a crear el modelo con la función *knn*.

```
#####################################################################
# KNN
#####################################################################

train.mIngreso <- train$mIngreso
model.knn <- knn(train, test, train.mIngreso)
```

2.6.2 Matriz de confusión

Una vez tenemos nuestro modelo, pasamos a ponerlo a prueba, para esto se realizó la matriz de confusión acorde. Una vez más se hizo uso del método *confusionMatrix*.

```
#####################################################################
# Matriz de confusión (KNN)
#####################################################################

test.mIngreso <- as.factor(test$mIngreso)
mcon.knn <- confusionMatrix(model.knn, test.mIngreso)
```

2.7 Reglas de clasificación

A los modelos anteriores se les sumó el modelo usando reglas de clasificación.

Para la creación de un modelo con éste método se decidió usar del paquete de **RWeka**, con éste paquete y la función *JRip*, fueron realizadas las reglas. Una vez más la entrada a la función fueron el conjunto de entrenamiento previamente creado y la formula definida por el método de ingreso.

Similar a lo ocurrido con la función *knn*, ciertas funciones deben ser de un tipo específico, por lo cual se realizó el cambio pertinente antes de aplicar la función *JRip*.

```
#####################################################################
# Buscando las reglas
#####################################################################

train$mIngreso <- as.factor(train$mIngreso)
rules <- JRip(formula = train$mIngreso ~ . , data = train)
```

2.7.1 Matriz de confusión

Para probar al modelo se usó una matriz de confusión, antes de poder aplicar ésta sin embargo, se probaron las reglas haciendo uso, una vez más de la función *predict*, así como del conjunto de prueba. Una vez se tuvo la predicción, sí se pasó a la realización de la matriz usando el mismo método usado tanto para como el árbol de decisión como el modelo hecho con k vecinos.

```
#####################################################################
# Predicción y matriz de confusión
#####################################################################

pred <- predict(rules, type="class")
test$mIngreso <- as.factor(test$mIngreso)
model.rules <- pred[1:length(test$mIngreso)]
mcom.rules <- confusionMatrix(model.rules, test$mIngreso)
```