Becas Crema 2.0

Deyban Pérez

March 5, 2016

Abstract

En la asignación anterior la compañía **Becas Crema** tomó los datos dados por **Control de Estudios** referentes a las renovaciones de beca con la meta de crear una vista minable. El objetivo fue completado y en esta oportunidad nos dispondremos a aplicar algunos algortimos para estimar el **modo de ingreso** de un estudiante que renueva la beca dada sus características.

Actividades Realizadas

1. Se cargó la vista minable creada en la asignación anterior **minable.csv**.

```
mydata = read.csv("minable.csv")
```

2. El siguiente paso fue el de selección de características, donde gracias al gran trabajo realizado previamente, todos los tipos de datos fueron categorizados y numerizados en caso de ser posible lo que nos permitió una fácil manipulación de las variables. Así mismo, se removieron aquellas columnas de tipo string y la fila cuyo individuo poseía un **modo de ingreso** tipo 1 (Convenios Nacionales o Internacionales), ya que sólo había una ocurrencia que no aportaba información.

```
##################################################
#Deleting no important and string features
mydata[,"cIdentidad"] = NULL
mydata[,"fNacimiento"] = NULL
mydata[,"jReprobadas"] = NULL
mydata[,"cDireccion"] = NULL
mydata[,"dHabitacion"] = NULL
mydata[,"oSolicitudes"] = NULL
mydata[,"aEconomica"] = NULL
mydata[,"rating"] = NULL
mydata[,"sugerencias"] = NULL
mydata[,"eCivil"] = NULL
mydata = mydata[-1,] #Deleting outlier
##################################################
```

3. Al hacer un análisis de los datos se pueden ver como las clases están desbalanceadas:

```
sum(mydata[,"mIngreso"] == 0)
```

```
## [1] 71
```

```
sum(mydata[,"mIngreso"] == 2)
```

```
## [1] 8
```

```
sum(mydata[,"mIngreso"] == 3)
```

```
## [1] 110
```

En la clase cero (0, OPSU) existen 71 ocurrencias, en la clase dos (2, Convenios) existen 8 ocurrencias y en la clase tres (3, Prueba Interna) existen 110 ocurrencias, dando un total de ciento noventa individuos (190). Dicho esto podemos apreciar el hecho de que las proporciones no están balanceadas. Dada esta situación previamente planteada la estrategia a adoptar fue la de hacer un **muestreo estratificado** a la hora de dividir el conjunto de datos total en los conjuntos de datos para **entrenamiento** (training) y **prueba** (testing). Para eso se tómo la probabilidad condicional de que un individuo sea elegido dado que ya se sabe que pertenece a la clase, cómo se muestra a continuación:

```
prob_0 = 1/sum(mydata[,"mIngreso"] == 0);
prob_2 = 1/sum(mydata[,"mIngreso"] == 2);
prob_3 = 1/sum(mydata[,"mIngreso"] == 3);
```

4. Hacer la separación del **testing** y el **training**, dejando un ochenta por ciento (80 %) de los datos para el **training** y el restante para el **testing**:

```
#####################################################
#Calculating probabilities for each element into dataset
prob_0 = 1/sum(mydata[,"mIngreso"] == 0);
prob_2 = 1/sum(mydata[,"mIngreso"] == 2);
prob_3 = 1/sum(mydata[,"mIngreso"] == 3);
#####################################################
#Allocating space for vector of probabilities
probabilities = seq(1,nrow(mydata),1)
#####################################################
aux0 = 0 #Constant for numebr 0
aux2 = 2 #Constant for number 2
aux3= 3 #Constant for number 3
#####################################################
#Filling probabilities for each position in data set
for (i in nrow(mydata))
{
  aux = mydata$mIngreso[i]

  if(aux0 == aux)
  {
    probabilities[i] = prob_0

  }else if(aux2 == aux)
  {
    probabilities[i] = prob_2

  }else if(aux3 == aux)
  {
    probabilities[i] = prob_3
  }

}
#####################################################
```

```
#Splitting data into training and testing sets
set.seed(777)
sub = sample(nrow(mydata), floor(nrow(mydata) * 0.8), prob = probabilities, replace = F)
training <- mydata[sub, ]
testing <- mydata[-sub, ]
#########################################################
```

Las proporciones quedaron de la siguiente manera para el **training**:

```
sum(training[,"mIngreso"] == 0)
```

```
## [1] 59
```

```
sum(training[,"mIngreso"] == 2)
```

```
## [1] 7
```

```
sum(training[,"mIngreso"] == 3)
```

```
## [1] 85
```

Y de la siguiente manera para el **testing**:

```
sum(testing[,"mIngreso"] == 0)
```

```
## [1] 12
```

```
sum(testing[,"mIngreso"] == 2)
```

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

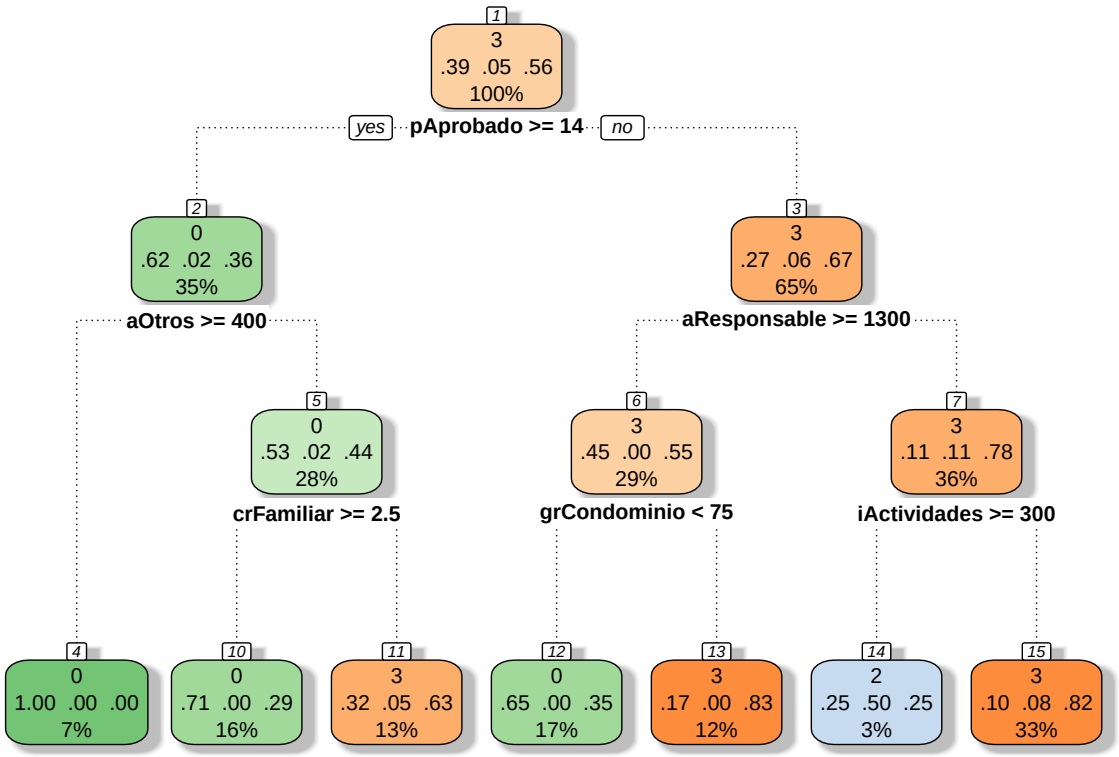
```
sum(testing[,"mIngreso"] == 3)
```

```
## [1] 25
```

Se puede apreciar como las proporciones se siguen manteniendo luego de la separación de los datos.

5. Se generó el modelo para el árbol de decisión usando la función **rpart**, con la **fórmula** (mIngreso ~.) que denota que se queire predecir la caracteŕistica **mIngreso** basándose en las demás características, el parámetro **method** (class) indica que queremos un modelo de clasificación, los parámetros de **control** (minsplit, cp, maxdepth) indican la cantidad mínima de individuos por nodo, el grado de brusquedad en los cambios y la profundidad de árbol respectivamente, estos parámetros fueron permutados hasta que se llegó al mejor resulatdo con los valores que se indican.

```
#########################################################
#Creating decission tree model to predict m Ingreso
tree = rpart(mIngreso ~ ., data = training, method = "class", control = rpart.control(minsplit = 10, cp
#Plotting model
fancyRpartPlot(tree)
```



6. Para evaluar el modelo se realizó una matriz de confusión y se calculó la tasa de error y de fallos evaluando la diagonal, tal como s emeustra a continuación:

```
################################################################################
#Creating confusion matrix tree
confusionMatrixTree = table(testing$mIngreso, predict(tree, newdata = testing,type = "class"))
#Visualizing Confusion matriz
confusionMatrixTree
```

```
##
##      0  2  3
##   0  7  0  5
##   2  0  0  1
##   3  9  1 15
```

```
################################################################################
#Calculating hit rate tree
hitRateTree = ((confusionMatrixTree[1,1] + confusionMatrixTree[2,2] + confusionMatrixTree[3,3]) / nrow(
#Visualizing hit rate tree
hitRateTree
```

```
## [1] 57.89474
```

```
################################################################################
#Calculating error rating tree
errorRateTree = 100 - hitRateTree
#Visualizing error rate
errorRateTree
```

```
## [1] 42.10526
```

```
##############################################################################
```

7. Se reacomodan los parámetro a ser pasados para el algoritmo de **K-Vecinos**, para esto se debió normalizar las columnas y eliminar las etiquetas de los conjuntos de **entrenamiento** (training) y **prueba** (training).

```
##############################################################################
# Beginning k nearest neighbor model
##############################################################################
#Extracting labels for traing and test
trainingLabels = training$mIngreso
testingLabels = testing$mIngreso
##############################################################################
#Deleting rows to be predicted
trainingClasificationRules = training
testingClasificationRules = testing
auxPCA = training
training[,"mIngreso"] = NULL
testing[,"mIngreso"] = NULL
##############################################################################
#Normalizing training and testing sets
training_norm = as.data.frame(lapply(training[1:ncol(training)], normalize))
testing_norm = as.data.frame(lapply(testing[1:ncol(testing)], normalize))
##############################################################################
```

8. Se generó el modelo de **K-Vecinos** haciendo uso de la función **knn**, pasando los conjutnos de prueba y entrenamiento **normalizados** y el parámetro **cl** que indica la vecindad a preguntar **14** tomando como medida la raíz cuadrada del numero total de individuos en el data set (189) que suele ser la medida que mejor se comporta teoricamente.

```
##############################################################################
#Generating knn model
mydata_pred <- knn(train = training_norm, test = testing_norm, cl = trainingLabels, k=14)
##############################################################################
```

9. Evaluando el modelo de **K-Vecinos** de igual manera que en **árboles de decisión** se utilizó una matriz de confusión y se calcularon las tasas de aciertos y fallos.

```
##############################################################################
#Creating confusion matrix
confusionMatrixK = table(testingLabels, mydata_pred)
#Visualizing Confusion matriz
confusionMatrixK
```

```
##              mydata_pred
## testingLabels  0  2  3
##             0  8  0  4
##             2  0  0  1
##             3  4  0 21
```

```
#############################################################################
#Calculating hit rate
hitRateK = ((confusionMatrixK[1,1] + confusionMatrixK[2,2] + confusionMatrixK[3,3]) / nrow(testing)) *1
#Visualizing hit rate
hitRateK
```

```
## [1] 76.31579
```

```
#############################################################################
#Calculating error rating
errorRateK = 100 - hitRateK
#Visualizing error rate
errorRateK
```

```
## [1] 23.68421
```

```
#############################################################################
```

10. Se generó el modelo de **Reglas de Clasificación** haciendo uso de la función **JRip** provista por la interfaz **RWeka**, para ello debimos transformar los daros como se muestra a continuación:

```
#############################################################################
# Beginning clasification rules
#############################################################################
#Transforming column into factor type
trainingClasificationRules$mIngreso = as.factor(trainingClasificationRules$mIngreso)
testingClasificationRules$mIngreso = as.factor(testingClasificationRules$mIngreso)
#############################################################################
```

11. Se generó el modelo de **Reglas de clasificación** como se muestra a continuación:

```
#############################################################################
#Generating Clasification Rules model
rules = JRip(formula = mIngreso ~ ., data = trainingClasificationRules)
#############################################################################
```

12.- De igual manera que en los modelos anteriores, se evaluó el modelo haciendo uso de una matriz de confusión, y se calcularon las tasas de aciertos y fallos, tal cómo se muestra a continuación:

```
#############################################################################
#Creating confusion matrix
confusionMatrixClasification = table(testingClasificationRules$mIngreso, predict(rules, newdata = testi
#Visualizing Confusion matriz
confusionMatrixClasification
```

```
##
##      0  2  3
##   0  5  1  6
##   2  0  0  1
##   3 10  1 14
```

```
#############################################################################################
#Calculating hit rate
hitRateClasification = ((confusionMatrixClasification[1,1] + confusionMatrixClasification[2,2] + confusi
#Visualizing hit rate
hitRateClasification
```

```
## [1] 50
```

```
#############################################################################################
#Calculating error rating
errorRateClasification = 100 - hitRateClasification
#Visualizing error rate
errorRateClasification
```

```
## [1] 50
```

```
#############################################################################################
#END
```

Comparación de Modelos

Para comparar los modelos podemos volver a visualizar las diferentes matrices de confusíon y las tasas de aciertos y fallas de los distintos modelos:

Árbol de decision

```
confusionMatrixTree
```

```
##
##      0  2  3
##   0  7  0  5
##   2  0  0  1
##   3  9  1 15
```

```
hitRateTree
```

```
## [1] 57.89474
```

```
errorRateTree
```

```
## [1] 42.10526
```

K-Vecinos

```
confusionMatrixK
```

```
##              mydata_pred
## testingLabels  0  2  3
##             0  8  0  4
##             2  0  0  1
##             3  4  0 21
```

```
hitRateK
```

```
## [1] 76.31579
```

```
errorRateK
```

```
## [1] 23.68421
```

Reglas de Clasificación

```
confusionMatrixClasification
```

```
##
##      0  2  3
##   0  5  1  6
##   2  0  0  1
##   3 10  1 14
```

```
hitRateClasification
```

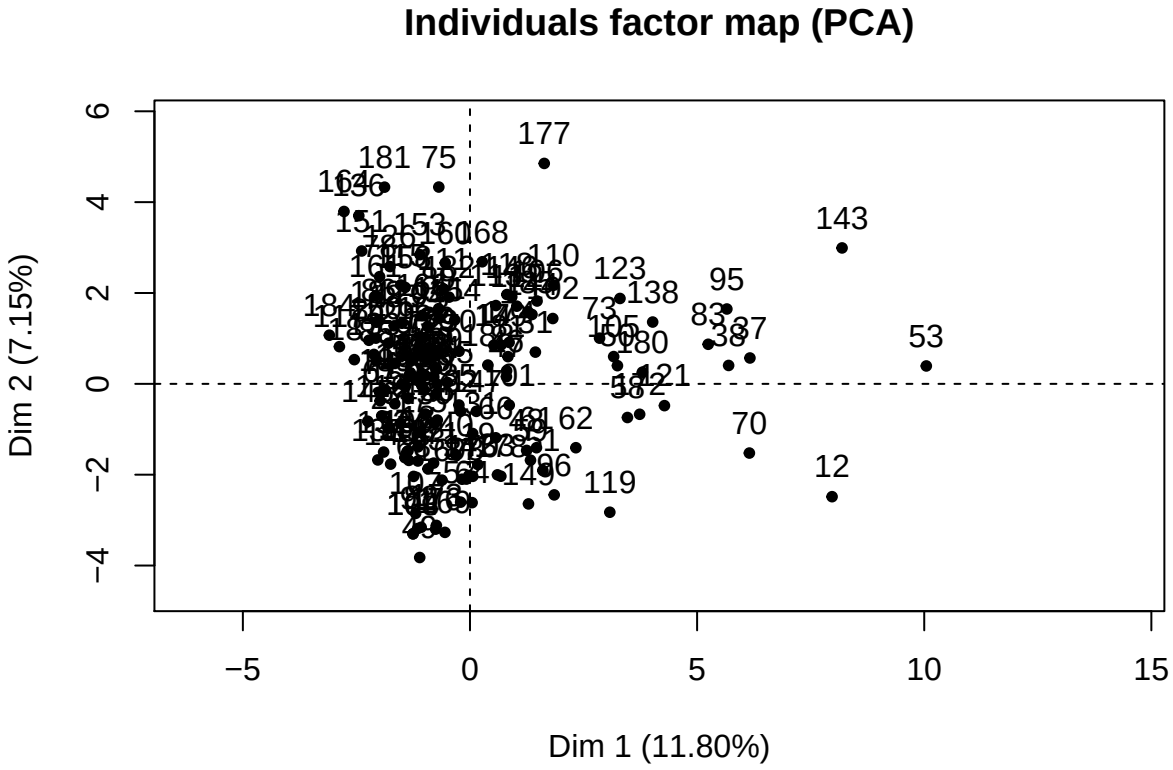
```
## [1] 50
```

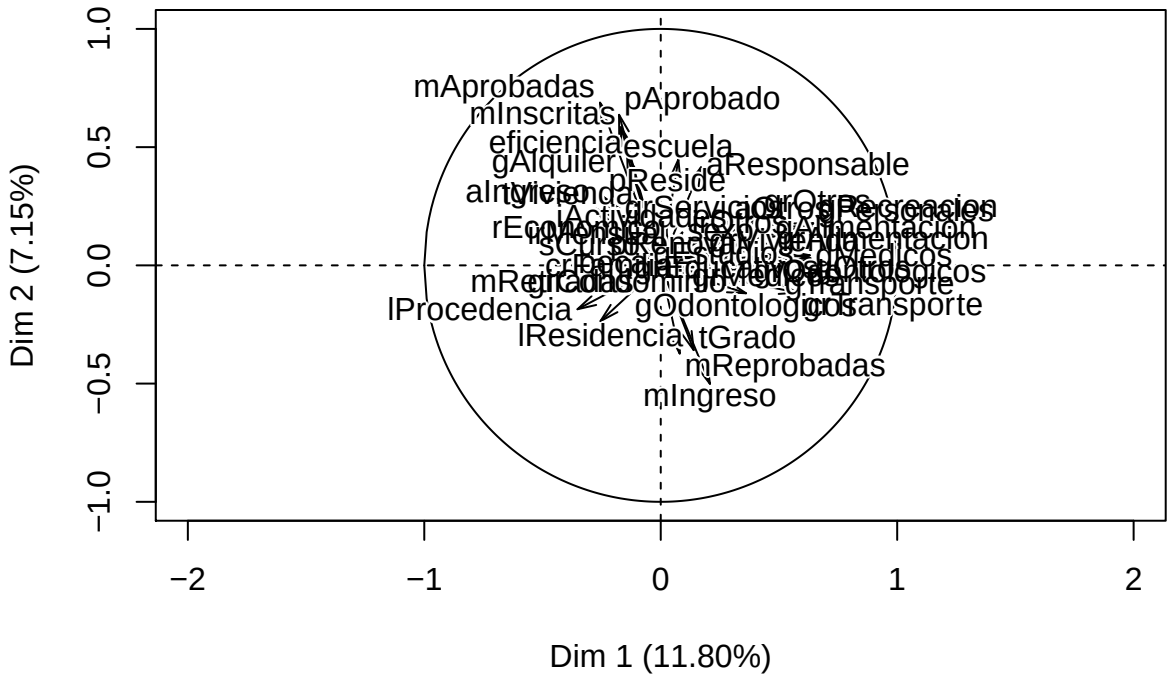
```
errorRateClasification
```

```
## [1] 50
```

Se puede ver como el modelo que presenta mejor porcentaje de acierto es **K-Vecinos**, seguido por **Árboles de decisión** y **Reglas de Clasificación** respectivamente. Aunque el modelo de **k-Vecinos** da un porcentaje de acierto por encima de setenta po ciento (70%), me parece que no e sun buen modelo debido a que las características no definen bein el problema y se tiene un conjunto de ocurrencias muy pequeño para el entrenamiento, debido a eso los otros dos algoritmos presentan un probe desempeño, ya que se hace difícil estimar, cómo se puede observar en el siguiente gráfico de **Anáilis de Componentes Principales** las características no representan de buena manera a todo el espacio.

```
PCA(auxPCA)
```






```
## **Results for the Principal Component Analysis (PCA)**
## The analysis was performed on 151 individuals, described by 43 variables
## *The results are available in the following objects:
##
```

```
##    name               description
## 1  "$eig"             "eigenvalues"
## 2  "$var"             "results for the variables"
## 3  "$var$coord"       "coord. for the variables"
## 4  "$var$cor"         "correlations variables - dimensions"
## 5  "$var$cos2"        "cos2 for the variables"
## 6  "$var$contrib"     "contributions of the variables"
## 7  "$ind"             "results for the individuals"
## 8  "$ind$coord"       "coord. for the individuals"
## 9  "$ind$cos2"        "cos2 for the individuals"
## 10 "$ind$contrib"     "contributions of the individuals"
## 11 "$call"            "summary statistics"
## 12 "$call$centre"     "mean of the variables"
## 13 "$call$ecart.type" "standard error of the variables"
## 14 "$call$row.w"      "weights for the individuals"
## 15 "$call$col.w"      "weights for the variables"
```