Becas Crema 2.0

Karina De Sousa

11 de marzo de 2016

Objectivo:

La Organización de Becas Crema ya ha obtenido una vista minable que permite su análisis para la detección de patrones subyacentes, usando dicho dataset, se requiere que compare el rendimiento de algoritmos de clasificación vistos en el curso que permitan etiquetar el modo de ingreso de la persona usando un subconjunto de las variables restantes seleccionado por usted.

Solución:

1. Cargamos los paquetes necesarios,

```
##  [1] "frbs"        "rpart"       "party"       "strucchange" "sandwich"
##  [6] "zoo"         "modeltools"  "stats4"      "mvtnorm"     "grid"
## [11] "gmodels"     "class"       "stats"       "graphics"    "grDevices"
## [16] "utils"       "datasets"    "methods"     "base"
```

2. Cargamos la vista minable.

```
> minable = read.csv(file = "./minable.csv", header = TRUE, sep = ",")  #Vista Minable original
```

3. La columna a predecir es “mIngreso”, que representa la modalidad de Ingreso a la universidad del estudiante, y puede tomar los valores,

- 0 (Asignado OPSU),
- 1 (Convenios Interinstitucionales (nacionales e internacionales)),
- 2 (Convenios Internos (Deportistas, artistas, hijos empleados docente y obreros, Samuel Robinson)),
- 3 (Prueba Interna y/o propedeútico).

Para esto tomaremos las primeras 20 columnas de la vista minable, exceptuando `fNacimiento, aIngreso, sCurso, tGrado, jReprobadas, pRenovar, beca`. Porque el resto de las columnas dan información sobre el estatus económico del becario y sus familiares o representantes.

```
> rmColumns = c("fNacimiento", "aIngreso", "sCurso", "tGrado", "jReprobadas",
+     "pRenovar", "beca", "mIngreso")
> columns = head(colnames(minable), n = 20)
>
> diff = columns[!(columns %in% rmColumns)]
>
> matrix = subset(x = minable, select = diff)  #subset para clasificacion
```

El dataframe `matrix` contiene las columnas que serán usadas en los distintos métodos de clasificación.

- **k vecinos más cercanos**

4. Para usar `knn` del paquete `class`, los valores deben estar estandarizados,

```
> matrixScale = subset(x = minable, select = diff)  #subset para clasificacion
> for (i in 2:length(matrixScale)) {
+     matrixScale[i] = scale(matrixScale[i])
+ }
> mIngresoScale = scale(minable$mIngreso)
```

5. Seleccionamos los valores de prueba y de entrenamiento,

```
> train <- minable$cIdentidad < 95
> test <- !train
>
> train.X <- matrixScale[train, ]
> test.X <- matrixScale[test, ]
> train.mIngreso <- minable$mIngreso[train]  #valor real de mIngreso en el set de
> # entrenamiento
> test.mIngreso <- minable$mIngreso[test]  #valor real de mIngreso en el set de pruebas
```

6. Usamos la función knn para predecir test.mIngreso, con k = 4 y vemos la calidad del modelo con una matriz de confusión.

```
> knn.pred <- knn(train = train.X, test = test.X, cl = train.mIngreso, k = 4)
>
> CrossTable(x = test.mIngreso, y = knn.pred, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  96
##
##
##                | knn.pred
## test.mIngreso |         0 |         3 | Row Total |
## --------------|-----------|-----------|-----------|
##             0 |        24 |        20 |        44 |
##               |     0.545 |     0.455 |     0.458 |
##               |     0.533 |     0.392 |           |
##               |     0.250 |     0.208 |           |
## --------------|-----------|-----------|-----------|
##             2 |         1 |         4 |         5 |
```

```
##                |     0.200 |     0.800 |     0.052 |
##                |     0.022 |     0.078 |           |
##                |     0.010 |     0.042 |           |
## --------------|-----------|-----------|-----------|
##              3 |        20 |        27 |        47 |
##                |     0.426 |     0.574 |     0.490 |
##                |     0.444 |     0.529 |           |
##                |     0.208 |     0.281 |           |
## --------------|-----------|-----------|-----------|
##   Column Total |        45 |        51 |        96 |
##                |     0.469 |     0.531 |           |
## --------------|-----------|-----------|-----------|
##
##
```

```
> tableC = table(knn.pred, test.mIngreso)
>
> # u = union(knn.pred, test.mIngreso) t = table(factor(knn.pred, u),
> # factor(test.mIngreso, u))
>
> # confusionMatrix(t)
```

En el conjunto de prueba seleccionado solo existen filas de las clases 0, 2 y 3. Cuando `k = 4`, todas son clasificadas como 0 ó 3, por lo que existe un error en 5 filas de 96 (0.052).

7. Ahora, variamos el valor de k para verificar si el modelo mejora o empeora.

```
> knn.pred <- knn(train = train.X, test = test.X, cl = train.mIngreso, k = 20)
>
> CrossTable(x = test.mIngreso, y = knn.pred, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  96
##
##
##                | knn.pred
## test.mIngreso |         3 | Row Total |
## --------------|-----------|-----------|
##              0 |        44 |        44 |
##                |     0.458 |           |
## --------------|-----------|-----------|
##              2 |         5 |         5 |
##                |     0.052 |           |
## --------------|-----------|-----------|
```

```
##             3 |        47 |        47 |
##               |     0.490 |           |
## --------------|-----------|-----------|
##  Column Total |        96 |        96 |
## --------------|-----------|-----------|
##
##
```

Con `k = 20`, todas las filas son clasificadas como 3, por lo que existe un error en 49 filas de 96 (0.51). Por lo tanto, hay error en un poco más de la mitad del total de filas.

```
> knn.pred <- knn(train = train.X, test = test.X, cl = train.mIngreso, k = 2)
>
> CrossTable(x = test.mIngreso, y = knn.pred, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  96
##
##
##                | knn.pred
## test.mIngreso |         0 | Row Total |
## --------------|-----------|-----------|
##             0 |        44 |        44 |
##               |     0.458 |           |
## --------------|-----------|-----------|
##             2 |         5 |         5 |
##               |     0.052 |           |
## --------------|-----------|-----------|
##             3 |        47 |        47 |
##               |     0.490 |           |
## --------------|-----------|-----------|
##  Column Total |        96 |        96 |
## --------------|-----------|-----------|
##
##
```

Con `k = 2`, todas las filas son clasificadas como 0, por lo que existe un error en 52 filas de 96 (0.54). Por lo tanto, hay error en más de la mitad del total de filas.

Vemos que al aumentar o disminuir el valor de k, el modelo clasifica todas las filas en una sola clase. Por lo que el mejor modelo se obtiene con k=4, a pesar de que existen algunas instancias (5) que no son clasificadas de forma correcta, lo cual es entendible porque la cantidad de filas de la clase 2 en el conjunto de prueba es muy pequeño con respecto a los 0s y 3s.

- **Árboles de decisión**

8. Usamos el dataframe `matrix`.

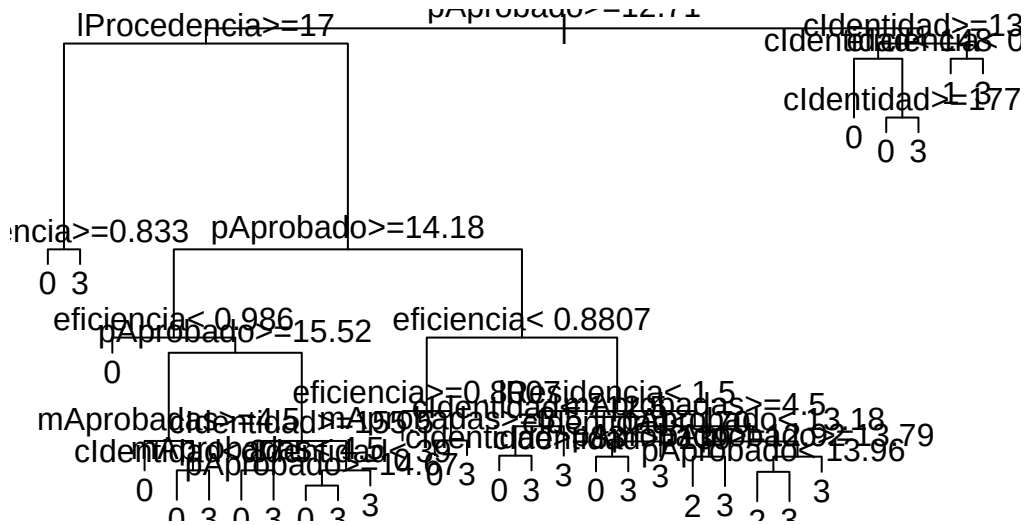
9. Creamos un objecto de tipo formula que le indica a la función que columnas debe usar para predecir `mIngreso`,

```
> formulaTree = diff[1]
> classColumn = "mIngreso"
> for (i in 2:length(diff)) {
+     if (diff[i] != classColumn) {
+         formulaTree = paste(formulaTree, diff[i], sep = " + ")
+     } else {
+         formulaTree = paste(classColumn, formulaTree, sep = " ~ ")
+     }
+ }
```

10. Creamos un árbol usando `rpart` y el parámetro `method = "class"`. El parámetro `minsplit` representa el número mínimo de observaciones que deben existir en un node antes de que se intente hacer una división, y `cp` es un parámetro de complejidad. Cualquier división que no disminuya la falta de presición en un factor de `cp` no será realizado.

En la esta prueba usaremos `minsplit = 5` y `cp = 0.001`.

```
> # Arbol usando rpart
> becas_tree <- rpart(formula = as.formula(formulaTree), data = matrix, method = "class",
+     control = rpart.control(minsplit = 5, cp = 0.001))
>
> # Graficamos el arbol
> plot(becas_tree)
> text(becas_tree)
```




```
> # Matriz de confusion
> predictionsProb = predict(becas_tree, type = "prob")
> predictionsClass = predict(becas_tree, type = "class")
>
> CrossTable(x = matrix$mIngreso, y = predictionsClass, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  190
##
##
##                | predictionsClass
## matrix$mIngreso |         0 |         1 |         2 |         3 | Row Total |
## ----------------|-----------|-----------|-----------|-----------|-----------|
##               0 |        60 |         0 |         0 |        11 |        71 |
##                 |     0.845 |     0.000 |     0.000 |     0.155 |     0.374 |
##                 |     0.896 |     0.000 |     0.000 |     0.094 |           |
```

```
##                 |     0.316 |     0.000 |     0.000 |     0.058 |           |
## ----------------|-----------|-----------|-----------|-----------|-----------|
##               1 |         0 |         1 |         0 |         0 |         1 |
##                 |     0.000 |     1.000 |     0.000 |     0.000 |     0.005 |
##                 |     0.000 |     0.500 |     0.000 |     0.000 |           |
##                 |     0.000 |     0.005 |     0.000 |     0.000 |           |
## ----------------|-----------|-----------|-----------|-----------|-----------|
##               2 |         3 |         1 |         4 |         0 |         8 |
##                 |     0.375 |     0.125 |     0.500 |     0.000 |     0.042 |
##                 |     0.045 |     0.500 |     1.000 |     0.000 |           |
##                 |     0.016 |     0.005 |     0.021 |     0.000 |           |
## ----------------|-----------|-----------|-----------|-----------|-----------|
##               3 |         4 |         0 |         0 |       106 |       110 |
##                 |     0.036 |     0.000 |     0.000 |     0.964 |     0.579 |
##                 |     0.060 |     0.000 |     0.000 |     0.906 |           |
##                 |     0.021 |     0.000 |     0.000 |     0.558 |           |
## ----------------|-----------|-----------|-----------|-----------|-----------|
##    Column Total |        67 |         2 |         4 |       117 |       190 |
##                 |     0.353 |     0.011 |     0.021 |     0.616 |           |
## ----------------|-----------|-----------|-----------|-----------|-----------|
##
##
```

```
> # print(becas_tree) summary(becas_tree)
> printcp(becas_tree)
```

```
##
## Classification tree:
## rpart(formula = as.formula(formulaTree), data = matrix, method = "class",
##     control = rpart.control(minsplit = 5, cp = 0.001))
##
## Variables actually used in tree construction:
## [1] cIdentidad   eficiencia   lProcedencia lResidencia  mAprobadas
## [6] pAprobado
##
## Root node error: 80/190 = 0.42105
##
## n= 190
##
##         CP nsplit rel error xerror    xstd
## 1 0.087500      0    1.0000 1.0000 0.08507
## 2 0.075000      2    0.8250 1.0375 0.08546
## 3 0.062500      3    0.7500 1.0000 0.08507
## 4 0.043750      4    0.6875 1.0000 0.08507
## 5 0.025000      6    0.6000 0.9500 0.08441
## 6 0.012500     13    0.4250 0.9625 0.08459
## 7 0.010714     22    0.3125 0.9500 0.08441
## 8 0.001000     29    0.2375 0.9750 0.08476
```

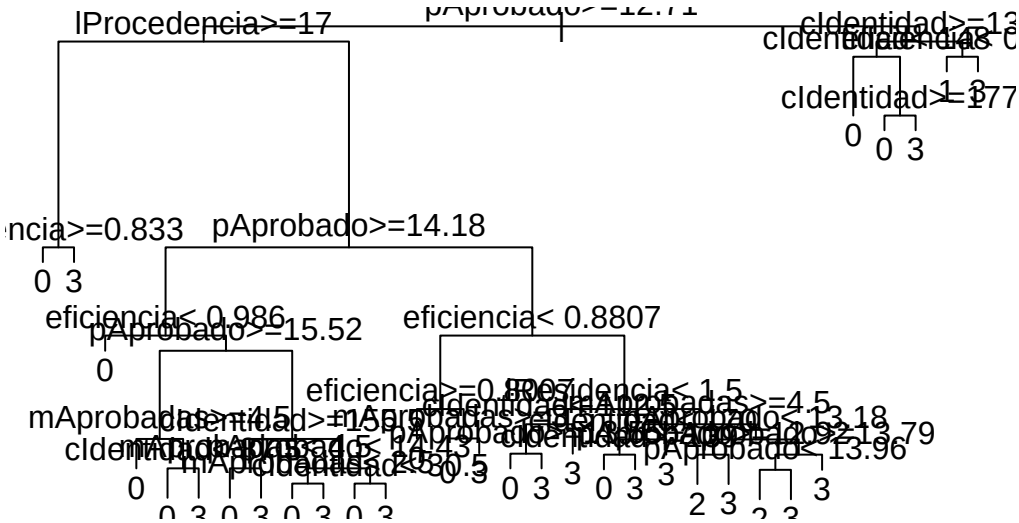
11. Viendo el resultado de `summary(becas_tree)`, las variables por orden de importancia en el modelo fueron `pAprobado`, `eficiencia`, `cIdentidad`, `lProcedencia`, `mAprobadas`, `lResidencia`, `mInscritas`, `mReprobadas`, `escuela` y `sexo`.

Mientras que las realmente usadas en la generación del árbol fueron, `cIdentidad, eficiencia, lProcedencia, lResidencia, mAprobadas` y `pAprobado`.

En este &aacuterbol vemos que 11 0s son clasificados como 3s, el único 1 es identificado de forma correcta, hay error en 4 2s y en 4 3s. En total, hubo errores en 19 filas de las 190 totales (0.1).

Vemos si el modelo mejora al eliminar las variables de menos importancia (`mReprobadas, escuela` y `sexo`),

```
> headnames = c("pAprobado", "eficiencia", "cIdentidad", "lProcedencia", "mAprobadas",
+     "lResidencia", "mInscritas", "mIngreso")
>
> # Nuevo dataframe
> matrixTree <- matrix[headnames]
>
> # Formula de clasificacion
> formulaTree = headnames[1]
> classColumn = "mIngreso"
> for (i in 2:length(headnames)) {
+     if (headnames[i] != classColumn) {
+         formulaTree = paste(formulaTree, headnames[i], sep = " + ")
+     } else {
+         formulaTree = paste(classColumn, formulaTree, sep = " ~ ")
+     }
+ }
>
> # Arbol usando rpart
> becas_tree <- rpart(formula = as.formula(formulaTree), data = matrixTree, method = "class",
+     control = rpart.control(minsplit = 5, cp = 0.001))
> plot(becas_tree)
> text(becas_tree)
```




```
> # Matriz de confusion
> predictionsProb = predict(becas_tree, type = "prob")
> predictionsClass = predict(becas_tree, type = "class")
> CrossTable(x = matrixTree$mIngreso, y = predictionsClass, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  190
##
##
##                      | predictionsClass
## matrixTree$mIngreso |         0 |         1 |         2 |         3 | Row Total |
## --------------------|-----------|-----------|-----------|-----------|-----------|
##                   0 |        62 |         0 |         0 |         9 |        71 |
##                     |     0.873 |     0.000 |     0.000 |     0.127 |     0.374 |
##                     |     0.886 |     0.000 |     0.000 |     0.079 |           |
##                     |     0.326 |     0.000 |     0.000 |     0.047 |           |
```

```
## --------------------|-----------|-----------|-----------|-----------|-----------|
##                   1 |         0 |         1 |         0 |         0 |         1 |
##                     |     0.000 |     1.000 |     0.000 |     0.000 |     0.005 |
##                     |     0.000 |     0.500 |     0.000 |     0.000 |           |
##                     |     0.000 |     0.005 |     0.000 |     0.000 |           |
## --------------------|-----------|-----------|-----------|-----------|-----------|
##                   2 |         3 |         1 |         4 |         0 |         8 |
##                     |     0.375 |     0.125 |     0.500 |     0.000 |     0.042 |
##                     |     0.043 |     0.500 |     1.000 |     0.000 |           |
##                     |     0.016 |     0.005 |     0.021 |     0.000 |           |
## --------------------|-----------|-----------|-----------|-----------|-----------|
##                   3 |         5 |         0 |         0 |       105 |       110 |
##                     |     0.045 |     0.000 |     0.000 |     0.955 |     0.579 |
##                     |     0.071 |     0.000 |     0.000 |     0.921 |           |
##                     |     0.026 |     0.000 |     0.000 |     0.553 |           |
## --------------------|-----------|-----------|-----------|-----------|-----------|
##        Column Total |        70 |         2 |         4 |       114 |       190 |
##                     |     0.368 |     0.011 |     0.021 |     0.600 |           |
## --------------------|-----------|-----------|-----------|-----------|-----------|
##
##
```

```
> # print(becas_tree) summary(becas_tree)
> printcp(becas_tree)
```

```
##
## Classification tree:
## rpart(formula = as.formula(formulaTree), data = matrixTree, method = "class",
##     control = rpart.control(minsplit = 5, cp = 0.001))
##
## Variables actually used in tree construction:
## [1] cIdentidad   eficiencia   lProcedencia lResidencia  mAprobadas
## [6] pAprobado
##
## Root node error: 80/190 = 0.42105
##
## n= 190
##
##         CP nsplit rel error xerror     xstd
## 1 0.087500      0    1.0000 1.0000 0.085070
## 2 0.075000      2    0.8250 1.1125 0.085978
## 3 0.062500      3    0.7500 1.0875 0.085844
## 4 0.043750      4    0.6875 1.0250 0.085340
## 5 0.025000      6    0.6000 0.9875 0.084920
## 6 0.012500     13    0.4250 1.0875 0.085844
## 7 0.010714     23    0.3000 1.0625 0.085672
## 8 0.001000     30    0.2250 1.1250 0.086031
```

12. Viendo el resultado de `summary(becas_tree)`, las variables por orden de importancia en el modelo fueron `pAprobado, eficiencia, cIdentidad, lProcedencia, mAprobadas, lResidencia` y `mInscritas`.

Mientras que las realmente usadas en la generación del árbol fueron, `cIdentidad, eficiencia, lProcedencia, lResidencia, mAprobadas` y `pAprobado`.

Con este modelo vemos que la clasificación de 1s y 2s se mantuvo igual, hubo error en 9 0s y en 5 3s. En total, hubo errores en 18 filas de las 190 totales (0.094), lo cual representa una mejora insignificante con respecto al modelo anterior.

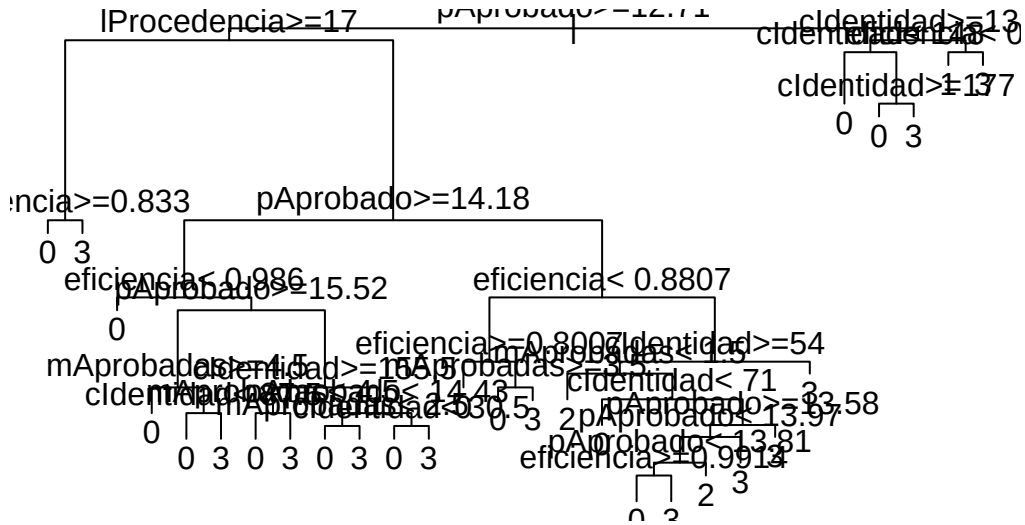
Todas las variables son usadas. Ahora vemos si el modelo empeora al eliminar las variables de menos importancia (`lResidencia` y `mInscritas`),

```
> headnames = c("pAprobado", "eficiencia", "cIdentidad", "lProcedencia", "mAprobadas",
+     "mIngreso")
>
> matrixTree <- matrix[headnames]
>
> formulaTree = headnames[1]
> classColumn = "mIngreso"
> for (i in 2:length(headnames)) {
+     if (headnames[i] != classColumn) {
+         formulaTree = paste(formulaTree, headnames[i], sep = " + ")
+     } else {
+         formulaTree = paste(classColumn, formulaTree, sep = " ~ ")
+     }
+ }
>
> # Arbol usando rpart
> becas_tree <- rpart(formula = as.formula(formulaTree), data = matrixTree, method = "class",
+     control = rpart.control(minsplit = 5, cp = 0.001))
> becas_tree
```

```
## n= 190
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##    1) root 190 80 3 (0.373684211 0.005263158 0.042105263 0.578947368)
##      2) pAprobado>=12.705 138 71 3 (0.471014493 0.000000000 0.043478261 0.485507246)
##        4) lProcedencia>=17 22  4 0 (0.818181818 0.000000000 0.000000000 0.181818182)
##          8) eficiencia>=0.833 19  2 0 (0.894736842 0.000000000 0.000000000 0.105263158) *
##          9) eficiencia< 0.833 3  1 3 (0.333333333 0.000000000 0.000000000 0.666666667) *
##        5) lProcedencia< 17 116 53 3 (0.405172414 0.000000000 0.051724138 0.543103448)
##         10) pAprobado>=14.1755 55 25 0 (0.545454545 0.000000000 0.018181818 0.436363636)
##           20) eficiencia< 0.986 7  0 0 (1.000000000 0.000000000 0.000000000 0.000000000) *
##           21) eficiencia>=0.986 48 24 3 (0.479166667 0.000000000 0.020833333 0.500000000)
##             42) pAprobado>=15.5225 21  8 0 (0.619047619 0.000000000 0.047619048 0.333333333)
##               84) mAprobadas>=4.5 11  2 0 (0.818181818 0.000000000 0.090909091 0.090909091) *
##               85) mAprobadas< 4.5 10  4 3 (0.400000000 0.000000000 0.000000000 0.600000000)
##                 170) cIdentidad< 87.5 4  1 0 (0.750000000 0.000000000 0.000000000 0.250000000) *
##                 171) cIdentidad>=87.5 6  1 3 (0.166666667 0.000000000 0.000000000 0.833333333) *
##             43) pAprobado< 15.5225 27 10 3 (0.370370370 0.000000000 0.000000000 0.629629630)
##               86) cIdentidad>=155.5 8  3 0 (0.625000000 0.000000000 0.000000000 0.375000000)
##                 172) mAprobadas< 4.5 4  0 0 (1.000000000 0.000000000 0.000000000 0.000000000) *
##                 173) mAprobadas>=4.5 4  1 3 (0.250000000 0.000000000 0.000000000 0.750000000) *
##               87) cIdentidad< 155.5 19  5 3 (0.263157895 0.000000000 0.000000000 0.736842105)
##                 174) pAprobado< 14.431 5  2 0 (0.600000000 0.000000000 0.000000000 0.400000000)
##                   348) mAprobadas< 2.5 2  0 0 (1.000000000 0.000000000 0.000000000 0.000000000) *
##                   349) mAprobadas>=2.5 3  1 3 (0.333333333 0.000000000 0.000000000 0.666666667) *
```

```
##                175) pAprobado>=14.431 14  2 3 (0.142857143 0.000000000 0.000000000 0.857142857)
##                  350) cIdentidad< 30.5 3  1 0 (0.666666667 0.000000000 0.000000000 0.333333333) *
##                  351) cIdentidad>=30.5 11  0 3 (0.000000000 0.000000000 0.000000000 1.000000000) *
##         11) pAprobado< 14.1755 61 22 3 (0.278688525 0.000000000 0.081967213 0.639344262)
##           22) eficiencia< 0.8807 12  4 0 (0.666666667 0.000000000 0.083333333 0.250000000)
##             44) eficiencia>=0.8007 7  0 0 (1.000000000 0.000000000 0.000000000 0.000000000) *
##             45) eficiencia< 0.8007 5  2 3 (0.200000000 0.000000000 0.200000000 0.600000000)
##               90) mAprobadas>=3.5 2  1 0 (0.500000000 0.000000000 0.500000000 0.000000000) *
##               91) mAprobadas< 3.5 3  0 3 (0.000000000 0.000000000 0.000000000 1.000000000) *
##           23) eficiencia>=0.8807 49 13 3 (0.183673469 0.000000000 0.081632653 0.734693878)
##             46) cIdentidad>=54 35 12 3 (0.228571429 0.000000000 0.114285714 0.657142857)
##               92) mAprobadas< 1.5 2  0 2 (0.000000000 0.000000000 1.000000000 0.000000000) *
##               93) mAprobadas>=1.5 33 10 3 (0.242424242 0.000000000 0.060606061 0.696969697)
##                 186) cIdentidad< 71 2  0 0 (1.000000000 0.000000000 0.000000000 0.000000000) *
##                 187) cIdentidad>=71 31  8 3 (0.193548387 0.000000000 0.064516129 0.741935484)
##                   374) pAprobado>=13.5825 15  7 3 (0.333333333 0.000000000 0.133333333 0.533333333)
##                     748) pAprobado< 13.975 8  4 0 (0.500000000 0.000000000 0.250000000 0.250000000)
##                      1496) pAprobado< 13.8055 5  2 0 (0.600000000 0.000000000 0.000000000 0.400000000)
##                        2992) eficiencia>=0.99145 3  0 0 (1.000000000 0.000000000 0.000000000 0.000000
##                        2993) eficiencia< 0.99145 2  0 3 (0.000000000 0.000000000 0.000000000 1.000000
##                      1497) pAprobado>=13.8055 3  1 2 (0.333333333 0.000000000 0.666666667 0.000000000)
##                     749) pAprobado>=13.975 7  1 3 (0.142857143 0.000000000 0.000000000 0.857142857) *
##                   375) pAprobado< 13.5825 16  1 3 (0.062500000 0.000000000 0.000000000 0.937500000) *
##             47) cIdentidad< 54 14  1 3 (0.071428571 0.000000000 0.000000000 0.928571429) *
##       3) pAprobado< 12.705 52  9 3 (0.115384615 0.019230769 0.038461538 0.826923077)
##         6) cIdentidad>=133 12  6 3 (0.416666667 0.000000000 0.083333333 0.500000000)
##          12) cIdentidad< 148 4  0 0 (1.000000000 0.000000000 0.000000000 0.000000000) *
##          13) cIdentidad>=148 8  2 3 (0.125000000 0.000000000 0.125000000 0.750000000)
##            26) cIdentidad>=177 2  1 0 (0.500000000 0.000000000 0.500000000 0.000000000) *
##            27) cIdentidad< 177 6  0 3 (0.000000000 0.000000000 0.000000000 1.000000000) *
##         7) cIdentidad< 133 40  3 3 (0.025000000 0.025000000 0.025000000 0.925000000)
##          14) eficiencia< 0.50195 2  1 1 (0.000000000 0.500000000 0.500000000 0.000000000) *
##          15) eficiencia>=0.50195 38  1 3 (0.026315789 0.000000000 0.000000000 0.973684211) *
```

```
> plot(becas_tree)
> text(becas_tree)
```




```
> predictionsProb = predict(becas_tree, type = "prob")
> predictionsClass = predict(becas_tree, type = "class")
>
> CrossTable(x = matrix$mIngreso, y = predictionsClass, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  190
##
##
##                  | predictionsClass
## matrix$mIngreso |         0 |         1 |         2 |         3 | Row Total |
## ----------------|-----------|-----------|-----------|-----------|-----------|
##               0 |        62 |         0 |         1 |         8 |        71 |
##                 |     0.873 |     0.000 |     0.014 |     0.113 |     0.374 |
##                 |     0.886 |     0.000 |     0.200 |     0.071 |           |
```

```
##                 |     0.326 |     0.000 |     0.005 |     0.042 |           |
## ----------------|-----------|-----------|-----------|-----------|-----------|
##               1 |         0 |         1 |         0 |         0 |         1 |
##                 |     0.000 |     1.000 |     0.000 |     0.000 |     0.005 |
##                 |     0.000 |     0.500 |     0.000 |     0.000 |           |
##                 |     0.000 |     0.005 |     0.000 |     0.000 |           |
## ----------------|-----------|-----------|-----------|-----------|-----------|
##               2 |         3 |         1 |         4 |         0 |         8 |
##                 |     0.375 |     0.125 |     0.500 |     0.000 |     0.042 |
##                 |     0.043 |     0.500 |     0.800 |     0.000 |           |
##                 |     0.016 |     0.005 |     0.021 |     0.000 |           |
## ----------------|-----------|-----------|-----------|-----------|-----------|
##               3 |         5 |         0 |         0 |       105 |       110 |
##                 |     0.045 |     0.000 |     0.000 |     0.955 |     0.579 |
##                 |     0.071 |     0.000 |     0.000 |     0.929 |           |
##                 |     0.026 |     0.000 |     0.000 |     0.553 |           |
## ----------------|-----------|-----------|-----------|-----------|-----------|
##    Column Total |        70 |         2 |         5 |       113 |       190 |
##                 |     0.368 |     0.011 |     0.026 |     0.595 |           |
## ----------------|-----------|-----------|-----------|-----------|-----------|
##
##
```

```
> # print(becas_tree) summary(becas_tree)
> printcp(becas_tree)
```

```
##
## Classification tree:
## rpart(formula = as.formula(formulaTree), data = matrixTree, method = "class",
##     control = rpart.control(minsplit = 5, cp = 0.001))
##
## Variables actually used in tree construction:
## [1] cIdentidad   eficiencia   lProcedencia mAprobadas   pAprobado
##
## Root node error: 80/190 = 0.42105
##
## n= 190
##
##         CP nsplit rel error xerror     xstd
## 1 0.087500      0    1.0000 1.0000 0.085070
## 2 0.075000      2    0.8250 1.0500 0.085571
## 3 0.062500      3    0.7500 1.0000 0.085070
## 4 0.043750      4    0.6875 1.0000 0.085070
## 5 0.025000      6    0.6000 0.9750 0.084760
## 6 0.016667     13    0.4250 0.8750 0.083114
## 7 0.012500     16    0.3750 0.8625 0.082861
## 8 0.001000     27    0.2250 0.9125 0.083808
```

13. Viendo el resultado de `summary(becas_tree)`, las variables por orden de importancia en el modelo fueron `eficiencia, pAprobado, cIdentidad, lProcedencia` y `mAprobadas`.

Mientras que las realmente usadas en la generación del árbol fueron, `cIdentidad, eficiencia, lProcedencia, mAprobadas` y `pAprobado`.

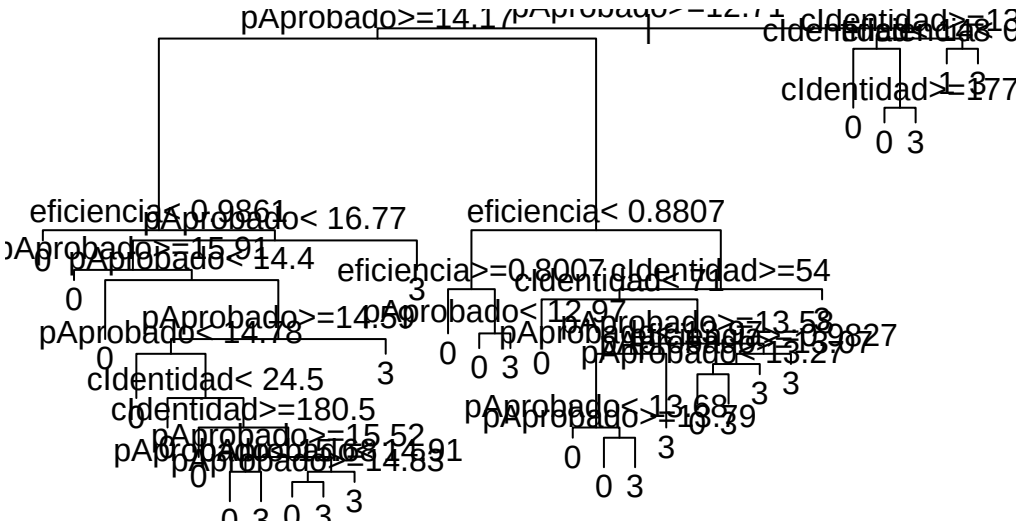
La calidad del modelo es la misma que la del anterior. Veamos que pasa al usar las 3 variables má importantes del modelo (`eficiencia, pAprobado` y `cIdentidad`),

```
> headnames = c("pAprobado", "eficiencia", "cIdentidad", "mIngreso")
>
> matrixTree <- matrix[headnames]
>
> formulaTree = headnames[1]
> classColumn = "mIngreso"
> for (i in 2:length(headnames)) {
+     if (headnames[i] != classColumn) {
+         formulaTree = paste(formulaTree, headnames[i], sep = " + ")
+     } else {
+         formulaTree = paste(classColumn, formulaTree, sep = " ~ ")
+     }
+ }
>
> # Arbol usando rpart
> becas_tree <- rpart(formula = as.formula(formulaTree), data = matrixTree, method = "class",
+     control = rpart.control(minsplit = 5, cp = 0.001))
> becas_tree
```

```
## n= 190
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##    1) root 190 80 3 (0.373684211 0.005263158 0.042105263 0.578947368)
##      2) pAprobado>=12.705 138 71 3 (0.471014493 0.000000000 0.043478261 0.485507246)
##        4) pAprobado>=14.1685 66 27 0 (0.590909091 0.000000000 0.015151515 0.393939394)
##          8) eficiencia< 0.9861 8  0 0 (1.000000000 0.000000000 0.000000000 0.000000000) *
##          9) eficiencia>=0.9861 58 27 0 (0.534482759 0.000000000 0.017241379 0.448275862)
##           18) pAprobado< 16.7725 55 24 0 (0.563636364 0.000000000 0.000000000 0.436363636)
##             36) pAprobado>=15.911 9  1 0 (0.888888889 0.000000000 0.000000000 0.111111111) *
##             37) pAprobado< 15.911 46 23 0 (0.500000000 0.000000000 0.000000000 0.500000000)
##               74) pAprobado< 14.404 6  1 0 (0.833333333 0.000000000 0.000000000 0.166666667) *
##               75) pAprobado>=14.404 40 18 3 (0.450000000 0.000000000 0.000000000 0.550000000)
##                150) pAprobado>=14.5935 33 16 0 (0.515151515 0.000000000 0.000000000 0.484848485)
##                  300) pAprobado< 14.775 4  0 0 (1.000000000 0.000000000 0.000000000 0.000000000) *
##                  301) pAprobado>=14.775 29 13 3 (0.448275862 0.000000000 0.000000000 0.551724138)
##                    602) cIdentidad< 24.5 2  0 0 (1.000000000 0.000000000 0.000000000 0.000000000) *
##                    603) cIdentidad>=24.5 27 11 3 (0.407407407 0.000000000 0.000000000 0.592592593)
##                     1206) cIdentidad>=180.5 2  0 0 (1.000000000 0.000000000 0.000000000 0.000000000)
##                     1207) cIdentidad< 180.5 25  9 3 (0.360000000 0.000000000 0.000000000 0.640000000)
##                       2414) pAprobado>=15.5225 11  5 0 (0.545454545 0.000000000 0.000000000 0.454545
##                         4828) pAprobado< 15.6805 5  1 0 (0.800000000 0.000000000 0.000000000 0.20000
##                         4829) pAprobado>=15.6805 6  2 3 (0.333333333 0.000000000 0.000000000 0.66666
##                       2415) pAprobado< 15.5225 14  3 3 (0.214285714 0.000000000 0.000000000 0.785714
##                         4830) pAprobado< 14.9075 5  2 3 (0.400000000 0.000000000 0.000000000 0.60000
##                           9660) pAprobado>=14.8275 3  1 0 (0.666666667 0.000000000 0.000000000 0.333
##                           9661) pAprobado< 14.8275 2  0 3 (0.000000000 0.000000000 0.000000000 1.000
##                         4831) pAprobado>=14.9075 9  1 3 (0.111111111 0.000000000 0.000000000 0.88888
##                151) pAprobado< 14.5935 7  1 3 (0.142857143 0.000000000 0.000000000 0.857142857) *
##           19) pAprobado>=16.7725 3  1 3 (0.000000000 0.000000000 0.333333333 0.666666667) *
```

```
##        5) pAprobado< 14.1685 72 31 3 (0.361111111 0.000000000 0.069444444 0.569444444)
##         10) eficiencia< 0.8807 15  6 0 (0.600000000 0.000000000 0.066666667 0.333333333)
##           20) eficiencia>=0.8007 7  0 0 (1.000000000 0.000000000 0.000000000 0.000000000) *
##           21) eficiencia< 0.8007 8  3 3 (0.250000000 0.000000000 0.125000000 0.625000000)
##             42) pAprobado< 12.9665 3  1 0 (0.666666667 0.000000000 0.000000000 0.333333333) *
##             43) pAprobado>=12.9665 5  1 3 (0.000000000 0.000000000 0.200000000 0.800000000) *
##         11) eficiencia>=0.8807 57 21 3 (0.298245614 0.000000000 0.070175439 0.631578947)
##           22) cIdentidad>=54 42 19 3 (0.357142857 0.000000000 0.095238095 0.547619048)
##             44) cIdentidad< 71 4  1 0 (0.750000000 0.000000000 0.250000000 0.000000000) *
##             45) cIdentidad>=71 38 15 3 (0.315789474 0.000000000 0.078947368 0.605263158)
##               90) pAprobado>=13.5825 18 10 0 (0.444444444 0.000000000 0.111111111 0.444444444)
##                180) pAprobado< 13.975 11  4 0 (0.636363636 0.000000000 0.181818182 0.181818182)
##                  360) pAprobado< 13.6815 3  0 0 (1.000000000 0.000000000 0.000000000 0.000000000) *
##                  361) pAprobado>=13.6815 8  4 0 (0.500000000 0.000000000 0.250000000 0.250000000)
##                    722) pAprobado>=13.7935 6  2 0 (0.666666667 0.000000000 0.333333333 0.000000000) *
##                    723) pAprobado< 13.7935 2  0 3 (0.000000000 0.000000000 0.000000000 1.000000000) *
##                181) pAprobado>=13.975 7  1 3 (0.142857143 0.000000000 0.000000000 0.857142857) *
##               91) pAprobado< 13.5825 20  5 3 (0.200000000 0.000000000 0.050000000 0.750000000)
##                182) eficiencia>=0.9827 12  4 3 (0.333333333 0.000000000 0.000000000 0.666666667)
##                  364) pAprobado>=13.0715 9  4 3 (0.444444444 0.000000000 0.000000000 0.555555556)
##                    728) pAprobado< 13.272 2  0 0 (1.000000000 0.000000000 0.000000000 0.000000000) *
##                    729) pAprobado>=13.272 7  2 3 (0.285714286 0.000000000 0.000000000 0.714285714) *
##                  365) pAprobado< 13.0715 3  0 3 (0.000000000 0.000000000 0.000000000 1.000000000) *
##                183) eficiencia< 0.9827 8  1 3 (0.000000000 0.000000000 0.125000000 0.875000000) *
##           23) cIdentidad< 54 15  2 3 (0.133333333 0.000000000 0.000000000 0.866666667) *
##      3) pAprobado< 12.705 52  9 3 (0.115384615 0.019230769 0.038461538 0.826923077)
##        6) cIdentidad>=133 12  6 3 (0.416666667 0.000000000 0.083333333 0.500000000)
##         12) cIdentidad< 148 4  0 0 (1.000000000 0.000000000 0.000000000 0.000000000) *
##         13) cIdentidad>=148 8  2 3 (0.125000000 0.000000000 0.125000000 0.750000000)
##           26) cIdentidad>=177 2  1 0 (0.500000000 0.000000000 0.500000000 0.000000000) *
##           27) cIdentidad< 177 6  0 3 (0.000000000 0.000000000 0.000000000 1.000000000) *
##        7) cIdentidad< 133 40  3 3 (0.025000000 0.025000000 0.025000000 0.925000000)
##         14) eficiencia< 0.50195 2  1 1 (0.000000000 0.500000000 0.500000000 0.000000000) *
##         15) eficiencia>=0.50195 38  1 3 (0.026315789 0.000000000 0.000000000 0.973684211) *
```

```
> plot(becas_tree)
> text(becas_tree)
```




```
> predictionsProb = predict(becas_tree, type = "prob")
> predictionsClass = predict(becas_tree, type = "class")
>
> CrossTable(x = matrix$mIngreso, y = predictionsClass, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  190
##
##
##                  | predictionsClass
## matrix$mIngreso |         0 |         1 |         3 | Row Total |
## ----------------|-----------|-----------|-----------|-----------|
##               0 |        61 |         0 |        10 |        71 |
##                 |     0.859 |     0.000 |     0.141 |     0.374 |
##                 |     0.871 |     0.000 |     0.085 |           |
```

```
##                 |     0.321 |     0.000 |     0.053 |           |
## ----------------|-----------|-----------|-----------|-----------|
##               1 |         0 |         1 |         0 |         1 |
##                 |     0.000 |     1.000 |     0.000 |     0.005 |
##                 |     0.000 |     0.500 |     0.000 |           |
##                 |     0.000 |     0.005 |     0.000 |           |
## ----------------|-----------|-----------|-----------|-----------|
##               2 |         4 |         1 |         3 |         8 |
##                 |     0.500 |     0.125 |     0.375 |     0.042 |
##                 |     0.057 |     0.500 |     0.025 |           |
##                 |     0.021 |     0.005 |     0.016 |           |
## ----------------|-----------|-----------|-----------|-----------|
##               3 |         5 |         0 |       105 |       110 |
##                 |     0.045 |     0.000 |     0.955 |     0.579 |
##                 |     0.071 |     0.000 |     0.890 |           |
##                 |     0.026 |     0.000 |     0.553 |           |
## ----------------|-----------|-----------|-----------|-----------|
##    Column Total |        70 |         2 |       118 |       190 |
##                 |     0.368 |     0.011 |     0.621 |           |
## ----------------|-----------|-----------|-----------|-----------|
##
##
```

```
> # print(becas_tree) summary(becas_tree)
> printcp(becas_tree)
```

```
##
## Classification tree:
## rpart(formula = as.formula(formulaTree), data = matrixTree, method = "class",
##     control = rpart.control(minsplit = 5, cp = 0.001))
##
## Variables actually used in tree construction:
## [1] cIdentidad eficiencia pAprobado
##
## Root node error: 80/190 = 0.42105
##
## n= 190
##
##          CP nsplit rel error xerror     xstd
## 1 0.0812500      0    1.0000 1.0000 0.085070
## 2 0.0500000      2    0.8375 0.9875 0.084920
## 3 0.0375000      3    0.7875 1.0375 0.085460
## 4 0.0250000      4    0.7500 1.0875 0.085844
## 5 0.0187500      6    0.7000 1.1500 0.086107
## 6 0.0125000     20    0.3875 1.0750 0.085763
## 7 0.0083333     25    0.3250 1.0625 0.085672
## 8 0.0062500     28    0.3000 1.0750 0.085763
## 9 0.0010000     30    0.2875 1.0875 0.085844
```

14. Vemos como el modelo empeora considerablemente con respecto a los anteriores, teniendo error en 23 filas (0.12). Por lo tanto, el mejor es el segundo, donde solo usamos las variables necesarias y eliminamos las que no eran usadas.

- **Reglas de clasificación**

15. Usamos `frbs`, para lo cual seleccionamos la data de entrenamiento y de prueba, donde la &uacute:ltima columna debe ser la que vamos a clasificar y la misma no puede contener ceros.

En esta primera prueba usamos `method.type <- "FRBCS.CHI"`, un sistema de clasificación basado en el método de Ishibuchi.

```
> # training with mIngresos test without mIngresos test real mIngresos
> train <- minable$cIdentidad < 95
> test <- !train
>
> train.X <- matrix[train, c(1:4, 6:13, 5)]
> test.X <- matrix[test, c(1:4, 6:13)]
>
> train.X[13] = train.X[13] + 1  #sumamos uno para eliminar los ceros
>
> test.mIngreso <- minable$mIngreso[test]  #valor real de mIngreso en el set de pruebas
>
> # define data range without mIngresos define method and control parameter
> range.data.input <- apply(matrix[, -ncol(matrix)], 2, range)
> method.type <- "FRBCS.W"
> control <- list(num.labels = 4, type.mf = "GAUSSIAN", type.tnorm = "MIN", type.snorm = "MAX",
+     type.implication.func = "ZADEH")
>
> # Learning step: Generate fuzzy model
> object.cls <- frbs.learn(train.X, range.data.input, method.type, control)
>
> # Predicting step: Predict newdata
> res.test <- predict(object.cls, test.X)
>
> res.test <- res.test - 1  #restamos el 1 que sumamos anteriormente
> CrossTable(x = test.mIngreso, y = res.test, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  96
##
##
##                | res.test
## test.mIngreso |         1 |         3 | Row Total |
## --------------|-----------|-----------|-----------|
##              0 |        35 |         9 |        44 |
##                |     0.795 |     0.205 |     0.458 |
##                |     0.565 |     0.265 |           |
##                |     0.365 |     0.094 |           |
```

```
## --------------|-----------|-----------|-----------|
##             2 |         3 |         2 |         5 |
##               |     0.600 |     0.400 |     0.052 |
##               |     0.048 |     0.059 |           |
##               |     0.031 |     0.021 |           |
## --------------|-----------|-----------|-----------|
##             3 |        24 |        23 |        47 |
##               |     0.511 |     0.489 |     0.490 |
##               |     0.387 |     0.676 |           |
##               |     0.250 |     0.240 |           |
## --------------|-----------|-----------|-----------|
##  Column Total |        62 |        34 |        96 |
##               |     0.646 |     0.354 |           |
## --------------|-----------|-----------|-----------|
##
##
```

16. En la matriz de confusión vemos que solo se clasifican de forma correcta 23 instancias del conjunto de prueba.

Ahora, usamos `method.type <- "FRBCS.CHI"`, un sistema de clasificación basado en el método de Chi.

```
> # training with mIngresos test without mIngresos test real mIngresos
> train <- minable$cIdentidad < 95
> test <- !train
>
> train.X <- matrix[train, c(1:4, 6:13, 5)]
> test.X <- matrix[test, c(1:4, 6:13)]
>
> train.X[13] = train.X[13] + 1
>
> test.mIngreso <- minable$mIngreso[test]  #valor real de mIngreso en el set de pruebas
>
> # define data range without mIngresos define method and control parameter
> range.data.input <- apply(matrix[, -ncol(matrix)], 2, range)
> method.type <- "FRBCS.CHI"
> control <- list(num.labels = 4, type.mf = "GAUSSIAN", type.tnorm = "MIN", type.snorm = "MAX",
+     type.implication.func = "ZADEH")
>
> # Learning step: Generate fuzzy model
> object.cls <- frbs.learn(train.X, range.data.input, method.type, control)
>
> # Predicting step: Predict newdata
> res.test <- predict(object.cls, test.X)
>
> res.test <- res.test - 1
> CrossTable(x = test.mIngreso, y = res.test, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
```

```
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  96
##
##
##                | res.test
## test.mIngreso |         1 |         3 | Row Total |
## --------------|-----------|-----------|-----------|
##             0 |        35 |         9 |        44 |
##               |     0.795 |     0.205 |     0.458 |
##               |     0.565 |     0.265 |           |
##               |     0.365 |     0.094 |           |
## --------------|-----------|-----------|-----------|
##             2 |         3 |         2 |         5 |
##               |     0.600 |     0.400 |     0.052 |
##               |     0.048 |     0.059 |           |
##               |     0.031 |     0.021 |           |
## --------------|-----------|-----------|-----------|
##             3 |        24 |        23 |        47 |
##               |     0.511 |     0.489 |     0.490 |
##               |     0.387 |     0.676 |           |
##               |     0.250 |     0.240 |           |
## --------------|-----------|-----------|-----------|
##  Column Total |        62 |        34 |        96 |
##               |     0.646 |     0.354 |           |
## --------------|-----------|-----------|-----------|
##
##
```

17. En la matriz de confusión vemos que solo se clasifican de forma correcta 23 instancias del conjunto de prueba, al igual que en el punto 16.

Ahora, usamos `method.type <- "FRBCS.W"` Y `type.mf <- BELL`,

```
> # training with mIngresos test without mIngresos test real mIngresos
> train <- minable$cIdentidad < 95
> test <- !train
>
> train.X <- matrix[train, c(1:4, 6:13, 5)]
> test.X <- matrix[test, c(1:4, 6:13)]
>
> train.X[13] = train.X[13] + 1
>
> test.mIngreso <- minable$mIngreso[test]  #valor real de mIngreso en el set de pruebas
>
> # define data range without mIngresos define method and control parameter
> range.data.input <- apply(matrix[, -ncol(matrix)], 2, range)
> method.type <- "FRBCS.W"
> control <- list(num.labels = 4, type.mf = "BELL", type.tnorm = "MIN", type.snorm = "MAX",
+     type.implication.func = "ZADEH")
```

```
>
> # Learning step: Generate fuzzy model
> object.cls <- frbs.learn(train.X, range.data.input, method.type, control)
>
> # Predicting step: Predict newdata
> res.test <- predict(object.cls, test.X)
>
> res.test <- res.test - 1
> CrossTable(x = test.mIngreso, y = res.test, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  96
##
##
##                | res.test
## test.mIngreso |         3 | Row Total |
## --------------|-----------|-----------|
##             0 |        44 |        44 |
##               |     0.458 |           |
## --------------|-----------|-----------|
##             2 |         5 |         5 |
##               |     0.052 |           |
## --------------|-----------|-----------|
##             3 |        47 |        47 |
##               |     0.490 |           |
## --------------|-----------|-----------|
##  Column Total |        96 |        96 |
## --------------|-----------|-----------|
##
##
```

18. En la matriz de confusión vemos que todas las instancias se asignan a la clase 3. Con lo cual tenemos 47 filas clasificadas de forma correcta.

Conclusión:

Basandonos en los resultados obtenidos, podemos concluir que la mejor clasificación se obtiene usando árboles de clasificación. Además este método permite visualizar de forma grafica los resultados y el proceso usado para clasificar cada instancia, lo cual lo hace más intuitivo y fácil de entender.