Becas Crema

Eric Bellet

11 de marzo de 2016

Introducción

El objetivo del script **BecasCrema.R** es utilizar una vista minable (minable.csv) con diversas variables asociadas a becas estudiantiles, para predecir utilizando algunas de estas, la modalidad de ingreso a la universidad por parte del estudiante.Para predecir se utilizará algoritmos de aprendizaje supervisado como **K-nearest-neighbours**, **arboles de decisión** y **reglas de clasificación**, para posteriormente evaluar cada uno de ellos y realizar comparaciones.

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 propedeutico.

Paquetes utilizados

library(sqldf)

library(dplyr)

library(class)

library(FactoMineR)

library(caret)

library(rpart)

library(rpart.plot)

library(RWeka)

library(pROC)

Carga del set de datos y preprocesamiento

Realizo la carga del set de datos y obtengo el gasto total e ingreso total del estudiante y del responsable económico para utilizarlos para predecir.

```
df <- read.csv("C:/Users/EricBellet/Desktop/AprendizajeSupervisado/data/minable.csv")
#Calculamos los ingresos personales del estudiante.
df$IngresosTotalPersonal <- df$aResponsable + df$iActividades + df$aOtros

#Calculamos los gastos personales del estudiante.
df$GastosTotalPersonal <- df$gAlimentacion + df$gTransporte + df$gMedicos +
  df$gOdontologicos + df$gAlquiler + df$gEstudios + df$gPersonales + df$gRecreacion + df$gOtros

#Calculamos los ingresos del responsable economico del estudiante.
df$IngresosTotalResponsable <- df$irMensual + df$irOtros
```

```r
#Calculamos los gastos del responsable economico del estudiante.
df$grOdontologicos <- as.numeric(df$grOdontologicos)
df$GastosTotalResponsable <- df$grAlimentacion + df$grTransporte + df$grMedicos +
  df$grOdontologicos + df$grEducativos + df$grVivienda + df$grServicios + df$grCondominio +
  df$grOtros
```

Posteriormente elimino las columnas cuya información no es númerica por lo tanto no nos sirve para los algoritmos que se utilizarán. Ordeno el dataframe que utilizaré.

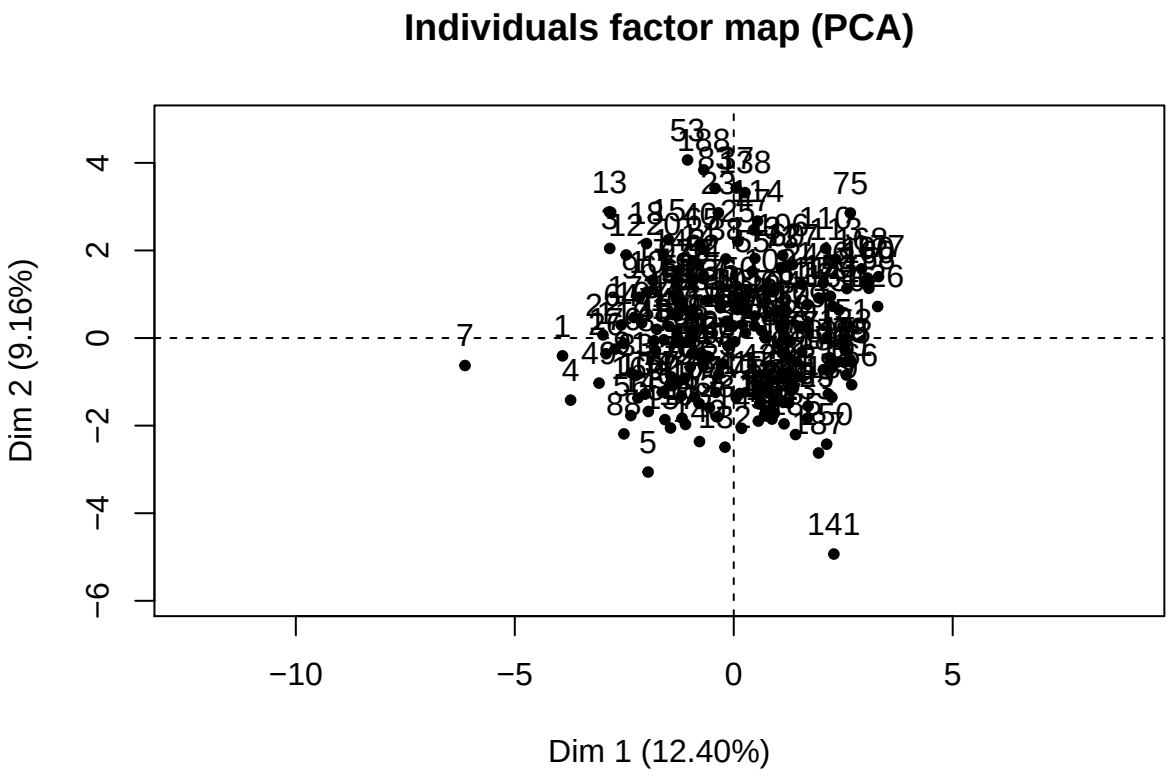
```r
#Eliminamos las columnas que no vamos a utilizar.
df$cIdentidad <- NULL
df$fNacimiento <- NULL
df$eCivil <- NULL
df$jReprobadas <- NULL
df$pReside <- NULL
df$dHabitacion <- NULL
df$cDireccion <- NULL
df$oSolicitudes <- NULL
df$aEconomica <- NULL
df$rating <- NULL
df$sugerencias <- NULL

#Seleccionamos variables predictoras.
df <- select(df, sexo, escuela, aIngreso, sCurso, tGrado, mInscritas,
             pAprobado, eficiencia, mAprobadas, mRetiradas, mReprobadas, pRenovar, beca,
             lProcedencia, lResidencia, tVivienda, rEconomico,
             crFamiliar, IngresosTotalPersonal, GastosTotalPersonal, IngresosTotalResponsable,
             GastosTotalResponsable,mIngreso)
```

Análisis exploratorio de los datos

Podemos observar que 71 estudiantes ingresaron por OPSU, 1 por convenio Interinstitucionales (nacionales e internacionales), 8 por convenios Internos(Deportistas,artistas, hijos empleados docente y obreros,Samuel Robinson) y 110 por prueba Interna y/o propedeutico.

```r
PCA <- PCA(df)
```



!

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

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

```
sum(df[,"mIngreso"] == 1)
```

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

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

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

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

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

Training y testing data

Se realizó un **muestreo estratificado** con la finalidad del que training entrene con todas las salidas y el testing pueda predecir todos los valores. Se obtuvó la probabilidad de cada valor de la modalidad de ingreso.

```r
#Obtengo los valores unicos de mIngreso.
valores <- unique(df$mIngreso)
totalvalores <- nrow(df)
probabilidad <- vector()
#Calculo la probabilidad de cada valor de mIngreso.
for (i in 1:length(valores)){
  probabilidad <- c(probabilidad, sum(df$mIngreso == valores[i]) / totalvalores)
}
asignarProb <- function(x){
  for (i in 1:length(valores)) {
    if (valores[i] == x){
       return(probabilidad[i])
    }

  }
}
#Obtengo un vector de probabilidades para cada valor de mIngreso.
probabilidades <- lapply(df$mIngreso, asignarProb)
probabilidades<-unlist(probabilidades)
```

Genero el **training** con un 70% de los datos y el **testing** con un 30%.

```r
#*********************************************************************************
#----------Genero un train y test data estratificado-------------------------------
#*********************************************************************************
set.seed(1)
sets <- sample(nrow(df), nrow(df)*0.7, prob=probabilidades, replace=F)
training <- df[sets,]
testing <- df[-sets,]
```

Proporción de **training**:

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

```
## [1] 43
```

```r
sum(training[,"mIngreso"] == 1)
```

```
## [1] 0
```

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

```
## [1] 2
```

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

```
## [1] 88
```

Proporción de **testing**:

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

```
## [1] 28
```

```r
sum(testing[,"mIngreso"] == 1)
```

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

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

```
## [1] 6
```

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

```
## [1] 22
```

K-nearest-neighbours

Se utilizó la biblioteca class para utilizar el algoritmo de **K-nearest-neighbours**, primero se realizó una normalización de los valores del dataset.

```r
#Genero las etiquetas.
cl <- training$mIngreso
#Normalizo el training y el testing para poder aplicar knn.
trainingN <-  as.data.frame(lapply(training, function (x) normalize(x)))
testingN <-  as.data.frame(lapply(testing, function (x) normalize(x)))
```

Genero el modelo, y para escoger el valor de K tome en cuenta los siguientes factores:

- Si K es muy pequeño el modelo será muy sentitivo a puntos que son atípicos o que son ruido (datos corruptos).
- Si K es muy grande, el modelo tiende a asignar siempre a la clase más grande.

Se utilizaron diferentes k hasta encontrar el que genera la mayor precisión general.

```r
knnModel<-knn(trainingN, testingN, cl, k = 3, prob=TRUE)
```

Podemos observar la **matriz de confusión**:

```r
matrizconfusion <- table(testing$mIngreso,knnModel,dnn=c("Valor Real", "Prediccion"))
print(matrizconfusion)
```

```
##           Prediccion
## Valor Real  0  2  3
##          0 22  1  5
##          1  0  0  1
##          2  0  0  6
##          3  2  0 20
```

En este contexto no es importante el error de precisión positiva y negativa (como en el contexto de predecir si una persona tiene o no cáncer), por lo tanto evaluaré la **precisión general**:

```
error <- (sum(knnModel != testing$mIngreso) /nrow(testing))
aciertoknn <- (1-error)*100
errorknn <- error*100
```

El porcentaje de **acierto** en general fue de:

```
print(aciertoknn)
```

```
## [1] 73.68421
```

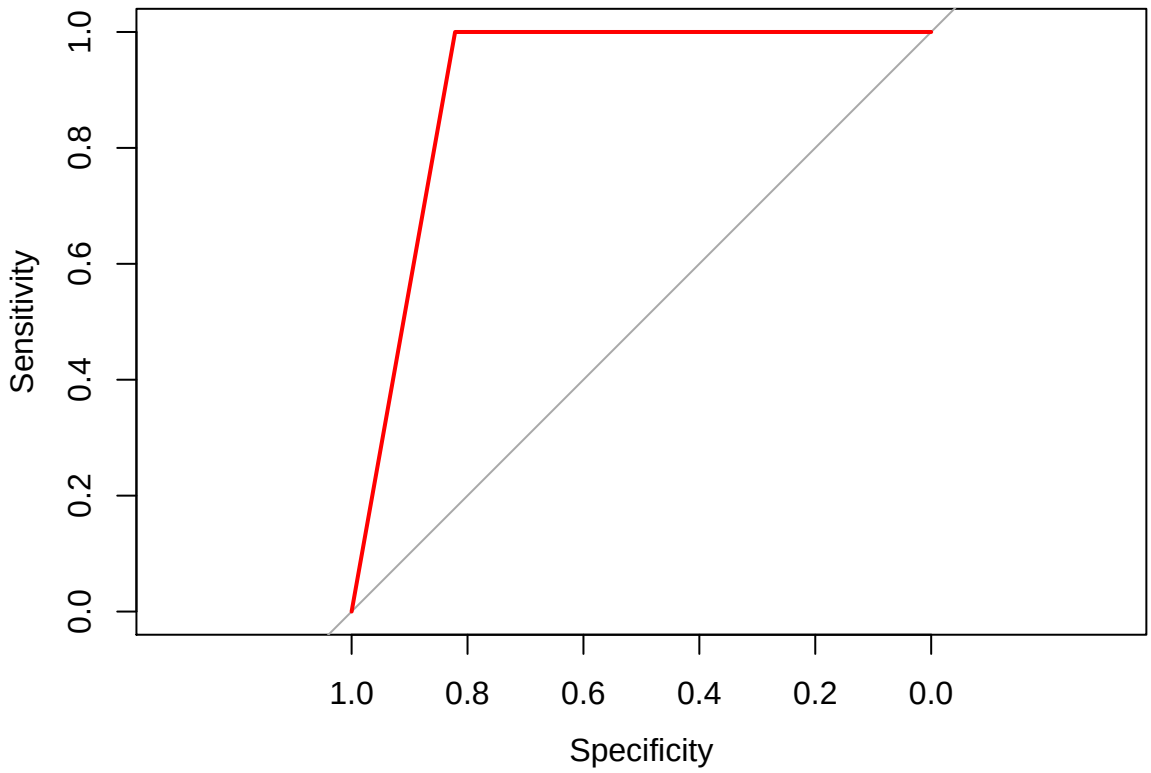
El porcentaje de **error** en general fue de:

```
print(errorknn)
```

```
## [1] 26.31579
```

Generamos la **curva de ROC**:

```
knnModel<-as.numeric(knnModel)
knnModelROC <- roc(testing$mIngreso, knnModel)
plot(knnModelROC,type="l",col="red")
```



```
##
## Call:
## roc.default(response = testing$mIngreso, predictor = knnModel)
##
## Data: knnModel in 28 controls (testing$mIngreso 0) < 1 cases (testing$mIngreso 1).
## Area under the curve: 0.9107
```

Arboles de Decisión

Se utilizó la biblioteca rpart para utilizar el algoritmo de **arboles de decisión**. Utilicé distintos valores de minsplit, cp, minbuckets y maxdepth, luego de jugar con todos estos valores utilicé los que me generarón mayor precisión general, que son ultimos valores de los arreglos.

```
minsplits <- c(1,1000,20,200,35)
cps <- c( 0.01,0.2,0.1,0.0001,0.0000000000001)
minbuckets <- c(50,5,10,50,300)
maxdepths <- c(30,5,10,20,30)
for (i in 1:length(minsplits)){
modelo <- rpart(mIngreso ~ ., data = training, method = "class",
                control = rpart.control(minsplit = minsplits[i],
                                        cp = cps[i],minbuckets=minbuckets[i],
                                        maxdepth = maxdepths[i]))
}
```

Realizo la predicción:

```
arbol <- predict(modelo, newdata = testing,type = "class")
```

Podemos observar la **matriz de confusión**:

```
matrizconfusion <- table(testing$mIngreso,arbol,dnn=c("Valor Real", "Prediccion"))
print(matrizconfusion)
```

```
##           Prediccion
## Valor Real  0  2  3
##          0 14  0 14
##          1  0  0  1
##          2  2  0  4
##          3  3  0 19
```

Evaluamos la precisión general del modelo.

```
error <- (sum(arbol != testing$mIngreso) /nrow(testing))
aciertoarbol <- (1-error)*100
errorarbol <- error*100
```

El porcentaje de **acierto** en general fue de:

```
print(aciertoarbol)
```

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

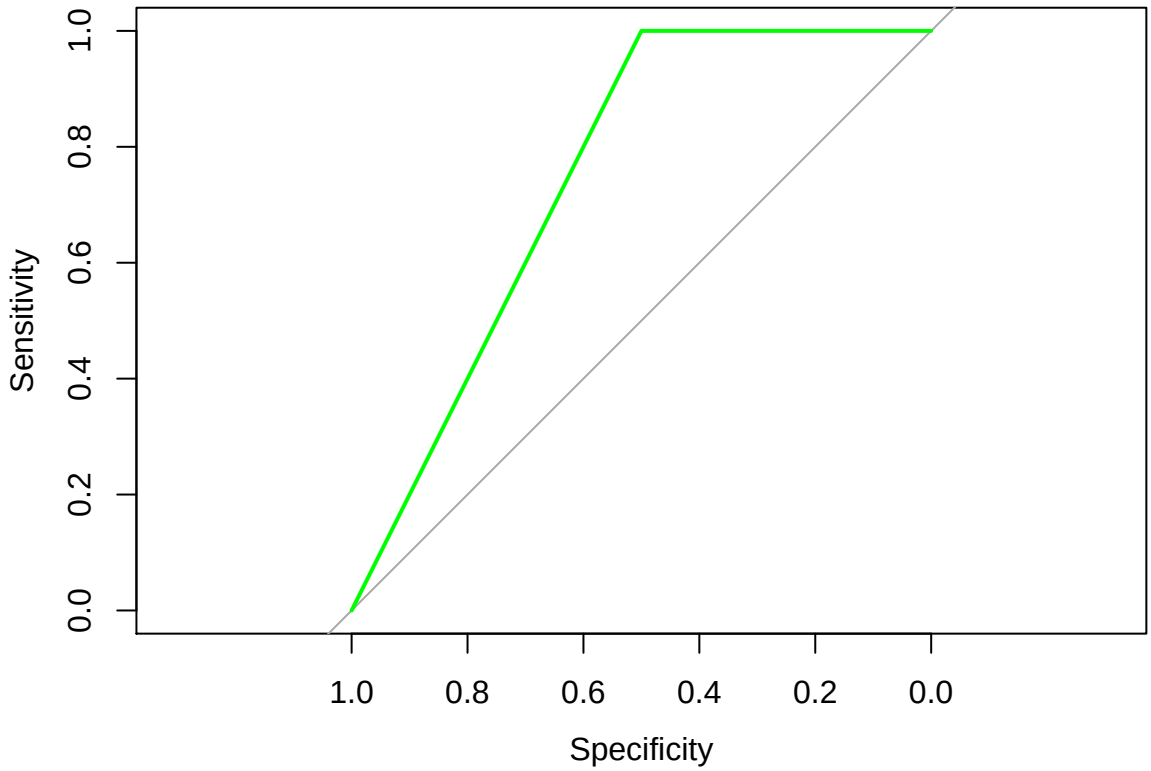
El porcentaje de **error** en general fue de:

```
print(errorarbol)
```

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

Generamos la **curva de ROC**:

```
arbol<-as.numeric(arbol)
arbolROC <- roc(testing$mIngreso, arbol)
plot(arbolROC,type="l",col="green")
```




```
##
## Call:
## roc.default(response = testing$mIngreso, predictor = arbol)
##
## Data: arbol in 28 controls (testing$mIngreso 0) < 1 cases (testing$mIngreso 1).
## Area under the curve: 0.75
```

Reglas de clasificación

Se utilizó la biblioteca RWeka para utilizar el algoritmo de **reglas de clasificación**. El algoritmo pide que la variable a predecir sea del tipo factor.

```
training$mIngreso = as.factor(training$mIngreso)
modelo <- JRip(mIngreso ~ ., training)
```

Realizo la predicción:

```
reglas <- predict(modelo, testing,type = "class")
```

Podemos observar la **matriz de confusión**:

```
matrizconfusion <- table(testing$mIngreso,reglas,dnn=c("Valor Real", "Prediccion"))
```

Evaluamos la precisión general del modelo.

```
error <- (sum(reglas != testing$mIngreso) /nrow(testing))
aciertoreglas <- (1-error)*100
errorreglas <- error*100
```

El porcentaje de **acierto** en general fue de:

```
print(aciertoreglas)
```

```
## [1] 40.35088
```

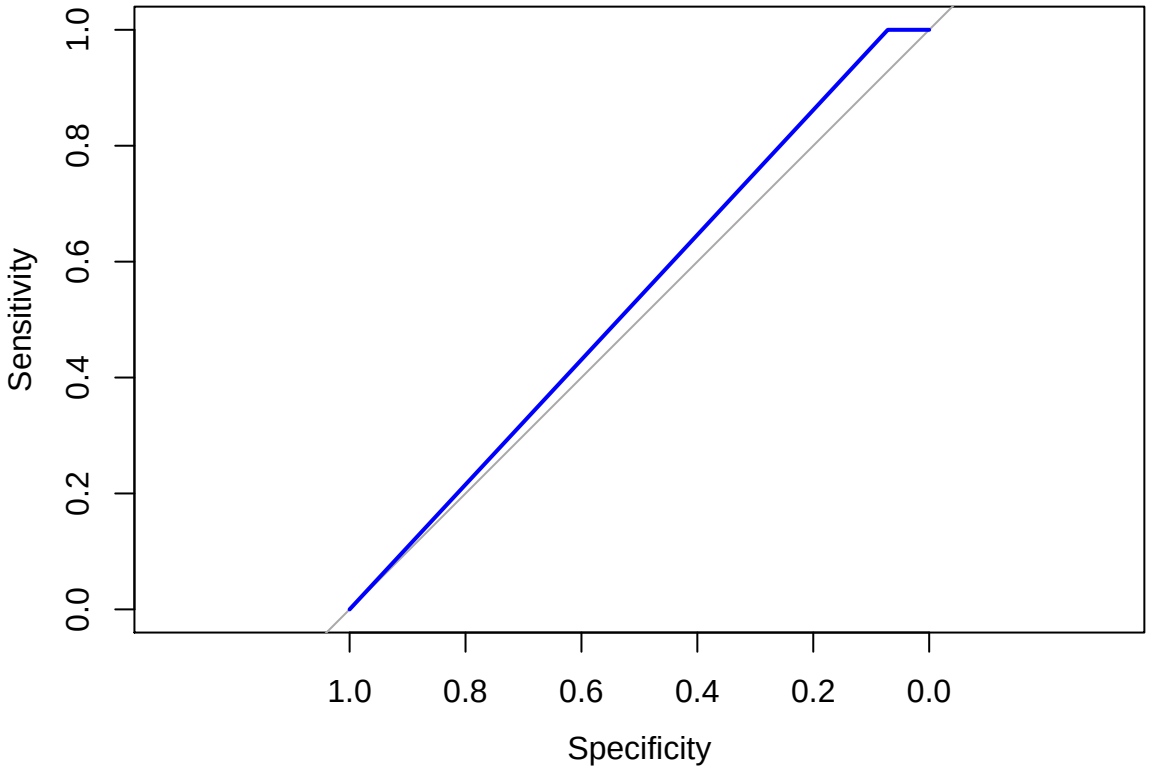
El porcentaje de **error** en general fue de:

```
print(errorreglas)
```

```
## [1] 59.64912
```

Generamos la **curva de ROC**:

```
reglas<-as.numeric(reglas)
reglasROC <- roc(testing$mIngreso, reglas)
plot(reglasROC,type="l",col="blue")
```




```
##
## Call:
## roc.default(response = testing$mIngreso, predictor = reglas)
##
## Data: reglas in 28 controls (testing$mIngreso 0) < 1 cases (testing$mIngreso 1).
## Area under the curve: 0.5357
```

Evaluamos los modelos

Podemos observar la precisión general de los 3 modelos:

```
## [1] "Precisión general de K-nearest-neighbours: 73.6842105263158"
```

```
## [1] "Precisión general de arboles de decisión: 57.8947368421053"
```

```
## [1] "Precisión general de reglas de clasificación: 40.3508771929825"
```

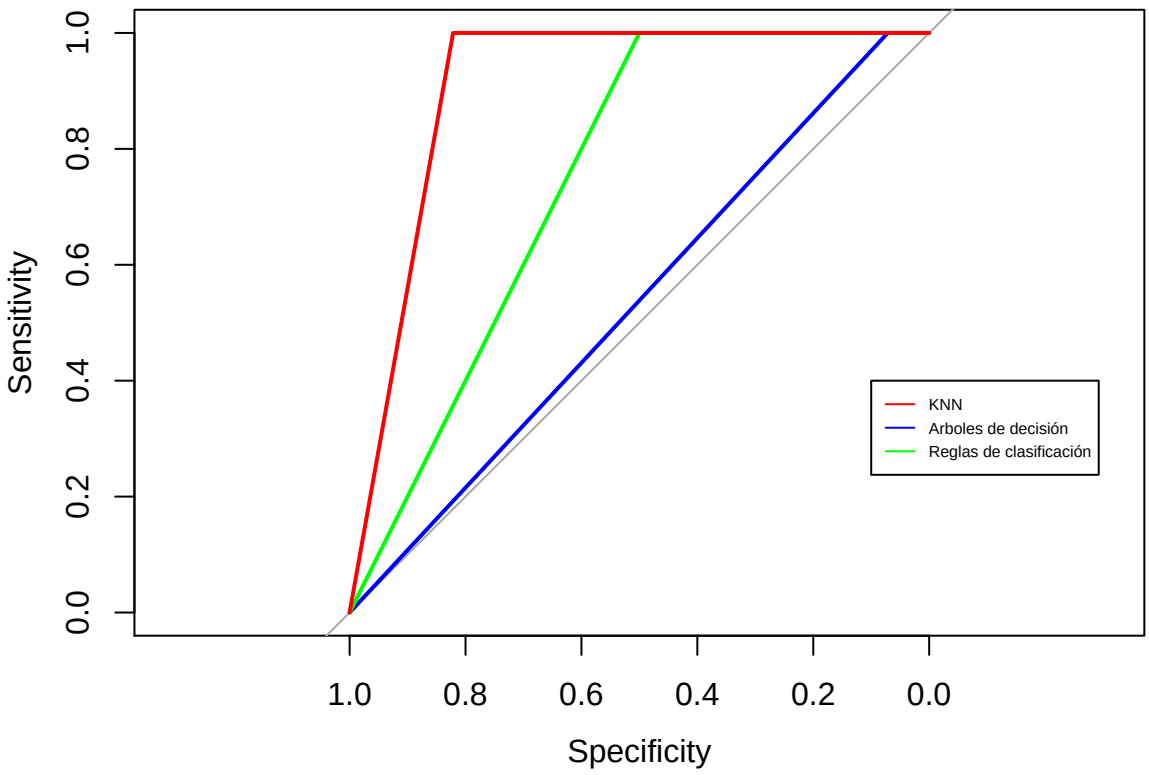
Observamos que el modelo de **K-nearest-neighbours** es el que posee la mayor precisión general.

Podemos observar la **curva de ROC** de los 3 modelos:

La sensibilidad es la probabilidad de clasificar correctamente a un individuo cuyo estado real es definido como positivo, respecto a la condición de prueba.

La especifidad es la probabilidad de clasificar correctamente a un individuo cuyo estado real es definido como negativo, respecto a la condición de prueba.

```
##
## Call:
## roc.default(response = testing$mIngreso, predictor = reglas)
##
## Data: reglas in 28 controls (testing$mIngreso 0) < 1 cases (testing$mIngreso 1).
## Area under the curve: 0.5357
```



Podemos concluir que el modelo de **K-nearest-neighbours** es el que mejor predice la modalidad de ingreso del dataset de **Becas Crema**.