Gradient Boosting Machine con métodos de "early stopping" OOV y CV. German Credit Data

Adolfo Sánchez Burón

Algoritmos empleados: Gradient Boosting Machine (GBM)

El **método de boosting**, junto con bagging y random forest, es lo que se conoce como **métodos de emsemble** o métodos combinados. Esto es, utilizan múltiples algoritmos de aprendizaje posibilitando una mejora de las predicciones que los obtenidos mediante algoritmos individuales, como son los árboles de decisión en este caso.

Su extensa utilización dentro del ámbito del machine learning se debe a que uno de los aspectos más negativos de los árboles de decisión es su alta varianza, es decir, son muy sensibles a variabilidad propia de los grupos de entrenamiento, pudiendo reportar resultados muy diferentes en función de las características de las muestras. Una manera de reducir esta varianza es emplear los **métodos de emsemble**. Sin embargo, estos métodos sufren, a su vez, de una problemática: son difícilmente interpretables, si los comparamos con los árboles de decisión.

La diferencia fundamental entre **random forest** (visto en otro post (https://www.ml2projects.com/post/churn-randomf)) y el **gradient boosting machine** (GBM) es que los primeros construyen un conjunto de árboles de decisión independientes, mientras que el GBM la construcción de árboles se realiza de manera sucesiva, es decir, cada árbol va mejorando al anterior.

El GBM se fundamenta en la idea de **training weak models** (entrenamiento de modelos débiles), esto es, mientras que random forest o bagging generan árboles muy profundos, GBM genera árboles que suelen tener entre 1 y 6 niveles de profundidad. Algunas ventajas de este procedimiento son:

- La velocidad en la ejecución del algoritmo.
- Posibilitar un learn slowly, aprendizaje lento y progresivo, paso a paso.
- Facilitar la detección del sobreajuste (overfitting) para el proceso cuando es detectado.

Uno de los aspectos más atractivos del GBM es su alta flexibilidad para ajustar (tuning) los parámetros. Estos son:

- **n.trees**: número de árboles óptimo para el mejor ajuste dle modelo.
- **Profundidad de los árboles**.
- **bag.fraction**: proporción de observaciones a ser muestreada en cada árbol
- **n.minobsinnode**: número mínimo de observaciones en cada nodo terminal.
- **interaction.depth**: nodos máximos por árbol.
- **shrinkage**: es el **learning rate** que controla la velocidad del procesamiento del algoritmo. Valores reducidos permiten controlar el sobreajuste, pero incrementan el tiempo de procesamiento para encontrar el resultado final.

Se recomienda leer:

Boehmke,B.C. Gradient Boosting Machines (http://uc-r.github.io/gbm_regression)

Gil,C. Árboles de decisión y métodos de ensemble (https://rpubs.com/Cristina_Gil/arboles_ensemble)

Hernández, F. Gradient Boost (https://fhernanb.github.io/libro_mod_pred/gradboost.html)

Ridgeway,G. Generalized Boosted Models: A guide to the gbm package (https://cran.r-project.org/web/packages/gbm/vignettes/gbm.pdf)

Características del caso

El caso empleado en este análisis es el 'German Credit Data', que puede descargarse el dataset original desde UCI (https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)). Este dataset ha sido previamente trabajado en cuanto a:

- análisis descriptivo
- limpieza de anomalías, missing y outliers
- peso predictivo de las variables mediante random forest
- discretización de las variables continuas para facilitar la interpretación posterior

Por lo que finalmente se emplea en este caso un dataset preparado para iniciar el análisis, que puede descargarse de GitHub (https://github.com/AdSan-R/MachineLearning_R/tree/main/dataset).

El objetivo del caso es predecir la probabilidad de que un determinado cliente puede incluir un crédito bancario. La explicación de esta conducta estará basada en toda una serie de variables predictoras que se explicarán posteriormente.

Proceso

1. Entorno

El primer punto tratará sobre la preparación del entorno, donde se mostrará la descarga de las librerías empleadas y la importación de datos.

2. Análisis descriptivo

Se mostrarán y explicarán las funciones empleadas en este paso, dividiéndolas en tres grupos: Análisis inicial, Tipología de datos y Análisis descriptivo (gráficos).

3. Preparación de la modelización

Particiones del dataset en dos grupos: training (70%) y test (30%)

4. Modelización

Por motivos didácticos, se dividirá la modelización de los dos algoritmos en una sucesión de pasos.

---

1. Entorno

1.1. Instalar librerías

```
library(dplyr)       # Manipulación de datos
library(knitr)       # Para formato de tablas
#library(ggplot2)     # Gráficos
#library(Information) # Exploración de datos con teoría del información
library(ROCR)        # Rendimiento del modelo y curva ROC
#library(lattice)     # Tratamiento de gráficos
library(caret)       # División de muestra, Clasificación y regresión
#library(rpart)       # Crear arboles de decisión
#library(rpart.plot)  # Gráfico del árbol
#library(randomForest)# Modelización mediante random forest
library(DataExplorer)# Análisis descriptivo con gráficos
library(gbm) #para algoritmo Gradient Boosting Machine
```

```
options(scipen=999)
#Desactiva la notación científica
```

1.2. Importar datos

Como el dataset ha sido peviamente trabajado para poder modelizar directamente, si deseas seguir este tutorial, lo puedes descargar de GitHub (https://github.com/AdSan-R/MachineLearning_R/tree/main/dataset).

```
df <- read.csv("CreditBank")
```

---

2. Análisis descriptivo

2.1. Análisis inicial

```
head(df) #ver los primeros 6 casos
```

```
##   X chk_ac_status_1 credit_history_3 duration_month_2 savings_ac_bond_6
## 1 1             A11           04.A34            00-06               A65
## 2 2             A12       03.A32.A33              42+               A61
## 3 3             A14           04.A34            06-12               A61
## 4 4             A11       03.A32.A33            36-42               A61
## 5 5             A11       03.A32.A33            12-24               A61
## 6 6             A14       03.A32.A33            30-36               A65
##   purpose_4 property_type_12 age_in_yrs_13 credit_amount_5 p_employment_since_7
## 1       A43             A121           60+          0-1400                  A75
## 2       A43             A121          0-25           5500+                  A73
## 3       A46             A121         45-50       1400-2500                  A74
## 4       A42             A122         40-45           5500+                  A74
## 5       A40             A124         50-60       4500-5500                  A73
## 6       A46             A124         30-35           5500+                  A73
##   housing_type_15 other_instalment_type_14 personal_status_9 foreign_worker_20
## 1            A152                    A143               A93              A201
## 2            A152                    A143               A92              A201
## 3            A152                    A143               A93              A201
## 4            A153                    A143               A93              A201
## 5            A153                    A143               A93              A201
## 6            A153                    A143               A93              A201
##   other_debtors_or_grantors_10 instalment_pct_8 good_bad_21
## 1                         A101                4        Good
## 2                         A101                2         Bad
## 3                         A101                2        Good
## 4                         A103                2        Good
## 5                         A101                3         Bad
## 6                         A101                2        Good
```

2.2. Tipología de datos

```
str(df) #mostrar la estructura del dataset y los tipos de variables
```

```
## 'data.frame':    1000 obs. of  17 variables:
##  $ X                           : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ chk_ac_status_1             : chr  "A11" "A12" "A14" "A11" ...
##  $ credit_history_3            : chr  "04.A34" "03.A32.A33" "04.A34" "03.A32.A33"
...
##  $ duration_month_2            : chr  "00-06" "42+" "06-12" "36-42" ...
##  $ savings_ac_bond_6           : chr  "A65" "A61" "A61" "A61" ...
##  $ purpose_4                   : chr  "A43" "A43" "A46" "A42" ...
##  $ property_type_12            : chr  "A121" "A121" "A121" "A122" ...
##  $ age_in_yrs_13               : chr  "60+" "0-25" "45-50" "40-45" ...
##  $ credit_amount_5             : chr  "0-1400" "5500+" "1400-2500" "5500+" ...
##  $ p_employment_since_7        : chr  "A75" "A73" "A74" "A74" ...
##  $ housing_type_15             : chr  "A152" "A152" "A152" "A153" ...
##  $ other_instalment_type_14    : chr  "A143" "A143" "A143" "A143" ...
##  $ personal_status_9           : chr  "A93" "A92" "A93" "A93" ...
##  $ foreign_worker_20           : chr  "A201" "A201" "A201" "A201" ...
##  $ other_debtors_or_grantors_10: chr  "A101" "A101" "A101" "A103" ...
##  $ instalment_pct_8            : int  4 2 2 2 3 2 3 2 2 4 ...
##  $ good_bad_21                 : chr  "Good" "Bad" "Good" "Good" ...
```

Puede observarse que todas son “chr”, esto es, “character”, por tanto, vamos a pasarlas a Factor. Además, instalment_pct_8 aparece como “entero” cuando es factor. También la transformamos.

```
df <- mutate_if(df, is.character, as.factor) #identifica todas las character y las pas
a a factores
#Sacamos la esructura

df$instalment_pct_8 <- as.factor(df$instalment_pct_8 )

str(df)
```

```
## 'data.frame':    1000 obs. of  17 variables:
##  $ X                          : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ chk_ac_status_1            : Factor w/ 4 levels "A11","A12","A13",..: 1 2 4 1 1
4 4 2 4 2 ...
##  $ credit_history_3           : Factor w/ 4 levels "01.A30","02.A31",..: 4 3 4 3 3
3 3 3 3 4 ...
##  $ duration_month_2           : Factor w/ 7 levels "00-06","06-12",..: 1 7 2 6 3 5
3 5 2 4 ...
##  $ savings_ac_bond_6          : Factor w/ 5 levels "A61","A62","A63",..: 5 1 1 1 1
5 3 1 4 1 ...
##  $ purpose_4                  : Factor w/ 10 levels "A40","A41","A410",..: 5 5 8 4
1 8 4 2 5 1 ...
##  $ property_type_12           : Factor w/ 4 levels "A121","A122",..: 1 1 1 2 4 4 2
3 1 3 ...
##  $ age_in_yrs_13              : Factor w/ 8 levels "0-25","25-30",..: 8 1 6 5 7 3
7 3 8 2 ...
##  $ credit_amount_5            : Factor w/ 6 levels "0-1400","1400-2500",..: 1 6 2
6 5 6 3 6 3 5 ...
##  $ p_employment_since_7       : Factor w/ 5 levels "A71","A72","A73",..: 5 3 4 4 3
3 5 3 4 1 ...
##  $ housing_type_15            : Factor w/ 3 levels "A151","A152",..: 2 2 2 3 3 3 2
1 2 2 ...
##  $ other_instalment_type_14   : Factor w/ 3 levels "A141","A142",..: 3 3 3 3 3 3 3
3 3 3 ...
##  $ personal_status_9          : Factor w/ 4 levels "A91","A92","A93",..: 3 2 3 3 3
3 3 3 1 4 ...
##  $ foreign_worker_20          : Factor w/ 2 levels "A201","A202": 1 1 1 1 1 1 1 1
1 1 ...
##  $ other_debtors_or_grantors_10: Factor w/ 3 levels "A101","A102",..: 1 1 1 3 1 1 1
1 1 1 ...
##  $ instalment_pct_8           : Factor w/ 4 levels "1","2","3","4": 4 2 2 2 3 2 3
2 2 4 ...
##  $ good_bad_21                : Factor w/ 2 levels "Bad","Good": 2 1 2 2 1 2 2 2 2
1 ...
```

Ahora se puede observar que todas las variables son de tipo “Factor”

Para los siguientes análisis:

1. Eliminamos a la variable X (número de cliente) del df.
2. Renombramos la variable good_bad_21 como “target”.

```
#Creamos la variable "target"
df$target <- as.factor(df$good_bad_21)

#Eliminamos la variable "good_bad_21" y eliminamos x
df <- select(df,-good_bad_21, -X)

str(df)
```

```
## 'data.frame':    1000 obs. of  16 variables:
##  $ chk_ac_status_1            : Factor w/ 4 levels "A11","A12","A13",..: 1 2 4 1 1
4 4 2 4 2 ...
##  $ credit_history_3           : Factor w/ 4 levels "01.A30","02.A31",..: 4 3 4 3 3
3 3 3 3 4 ...
##  $ duration_month_2           : Factor w/ 7 levels "00-06","06-12",..: 1 7 2 6 3 5
3 5 2 4 ...
##  $ savings_ac_bond_6          : Factor w/ 5 levels "A61","A62","A63",..: 5 1 1 1 1
5 3 1 4 1 ...
##  $ purpose_4                  : Factor w/ 10 levels "A40","A41","A410",..: 5 5 8 4
1 8 4 2 5 1 ...
##  $ property_type_12           : Factor w/ 4 levels "A121","A122",..: 1 1 1 2 4 4 2
3 1 3 ...
##  $ age_in_yrs_13              : Factor w/ 8 levels "0-25","25-30",..: 8 1 6 5 7 3
7 3 8 2 ...
##  $ credit_amount_5            : Factor w/ 6 levels "0-1400","1400-2500",..: 1 6 2
6 5 6 3 6 3 5 ...
##  $ p_employment_since_7       : Factor w/ 5 levels "A71","A72","A73",..: 5 3 4 4 3
3 5 3 4 1 ...
##  $ housing_type_15            : Factor w/ 3 levels "A151","A152",..: 2 2 2 3 3 3 2
1 2 2 ...
##  $ other_instalment_type_14   : Factor w/ 3 levels "A141","A142",..: 3 3 3 3 3 3 3
3 3 3 ...
##  $ personal_status_9          : Factor w/ 4 levels "A91","A92","A93",..: 3 2 3 3 3
3 3 3 1 4 ...
##  $ foreign_worker_20          : Factor w/ 2 levels "A201","A202": 1 1 1 1 1 1 1 1
1 1 ...
##  $ other_debtors_or_grantors_10: Factor w/ 3 levels "A101","A102",..: 1 1 1 3 1 1 1
1 1 1 ...
##  $ instalment_pct_8           : Factor w/ 4 levels "1","2","3","4": 4 2 2 2 3 2 3
2 2 4 ...
##  $ target                     : Factor w/ 2 levels "Bad","Good": 2 1 2 2 1 2 2 2 2
1 ...
```

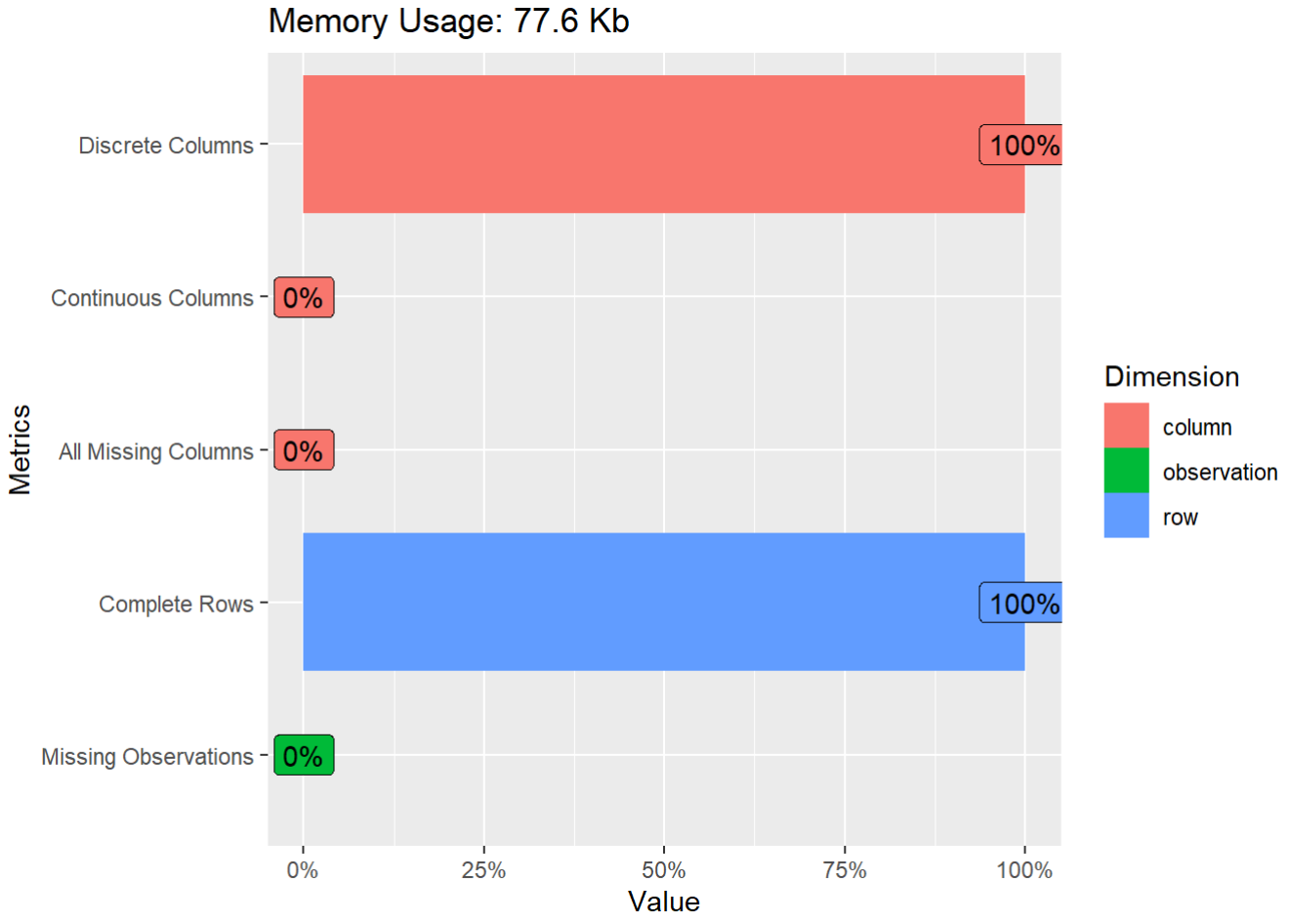
```
lapply(df,summary) #mostrar la distribución de frecuencias en cada categoría de todas
 las variables
```

```
## $chk_ac_status_1
## A11 A12 A13 A14
## 274 269  63 394
##
## $credit_history_3
##     01.A30     02.A31 03.A32.A33     04.A34
##         40         49        618        293
##
## $duration_month_2
## 00-06 06-12 12-24 24-30 30-36 36-42   42+
##    82   277   411    57    86    17    70
##
## $savings_ac_bond_6
## A61 A62 A63 A64 A65
## 603 103  63  48 183
##
## $purpose_4
##  A40  A41 A410  A42  A43  A44  A45  A46  A48  A49
##  234  103   12  181  280   12   22   50    9   97
##
## $property_type_12
## A121 A122 A123 A124
##  282  232  332  154
##
## $age_in_yrs_13
##  0-25 25-30 30-35 35-40 40-45 45-50 50-60   60+
##   190   221   177   138    88    73    68    45
##
## $credit_amount_5
##    0-1400 1400-2500 2500-3500 3500-4500 4500-5500     5500+
##       267        270        149         98         48       168
##
## $p_employment_since_7
## A71 A72 A73 A74 A75
##  62 172 339 174 253
##
## $housing_type_15
## A151 A152 A153
##  179  713  108
##
## $other_instalment_type_14
## A141 A142 A143
##  139   47  814
##
## $personal_status_9
## A91 A92 A93 A94
##  50 310 548  92
##
## $foreign_worker_20
## A201 A202
##  963   37
```

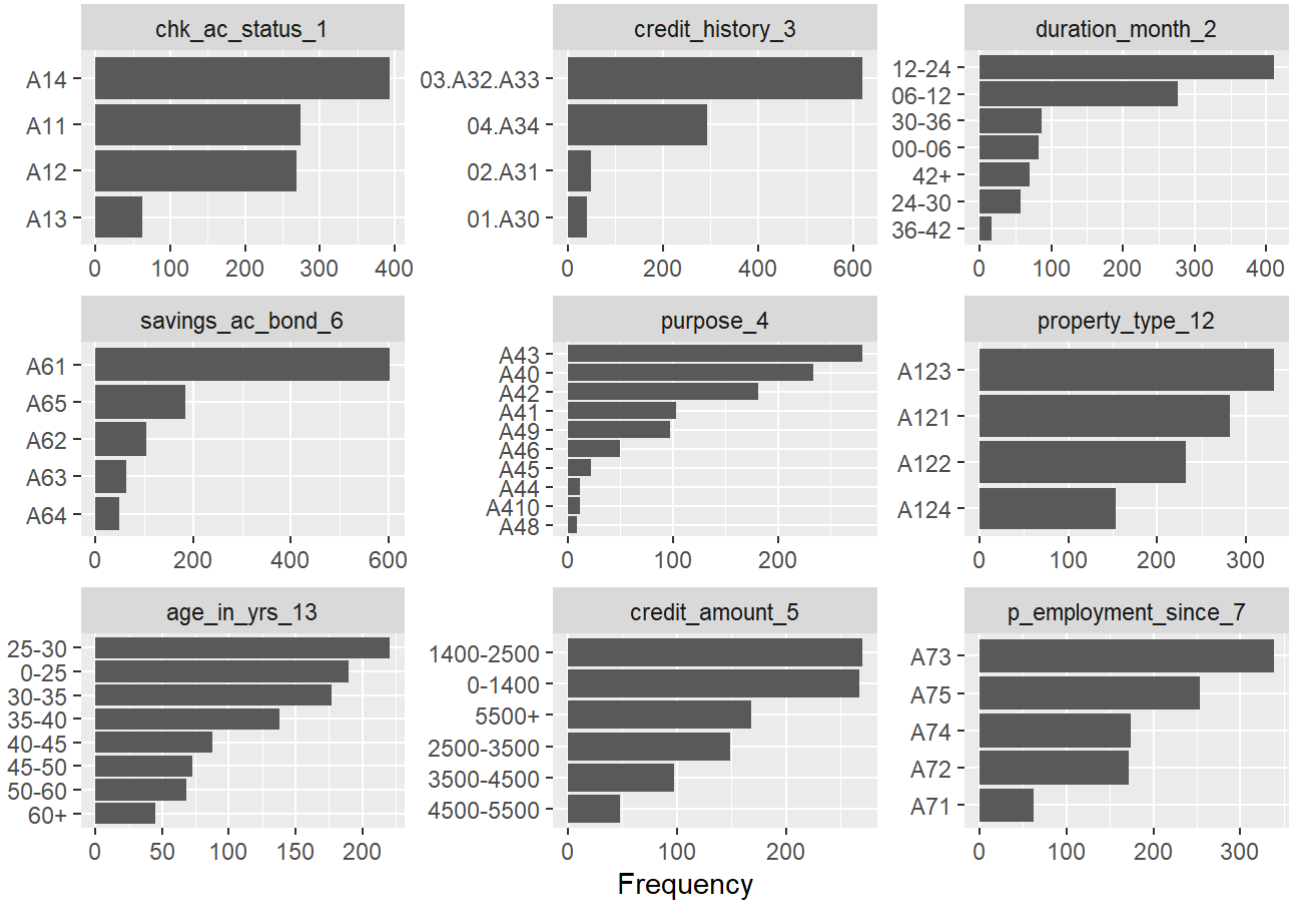
```
##
## $other_debtors_or_grantors_10
## A101 A102 A103
##  907   41   52
##
## $instalment_pct_8
##   1   2   3   4
## 136 231 157 476
##
## $target
##  Bad Good
##  300  700
```

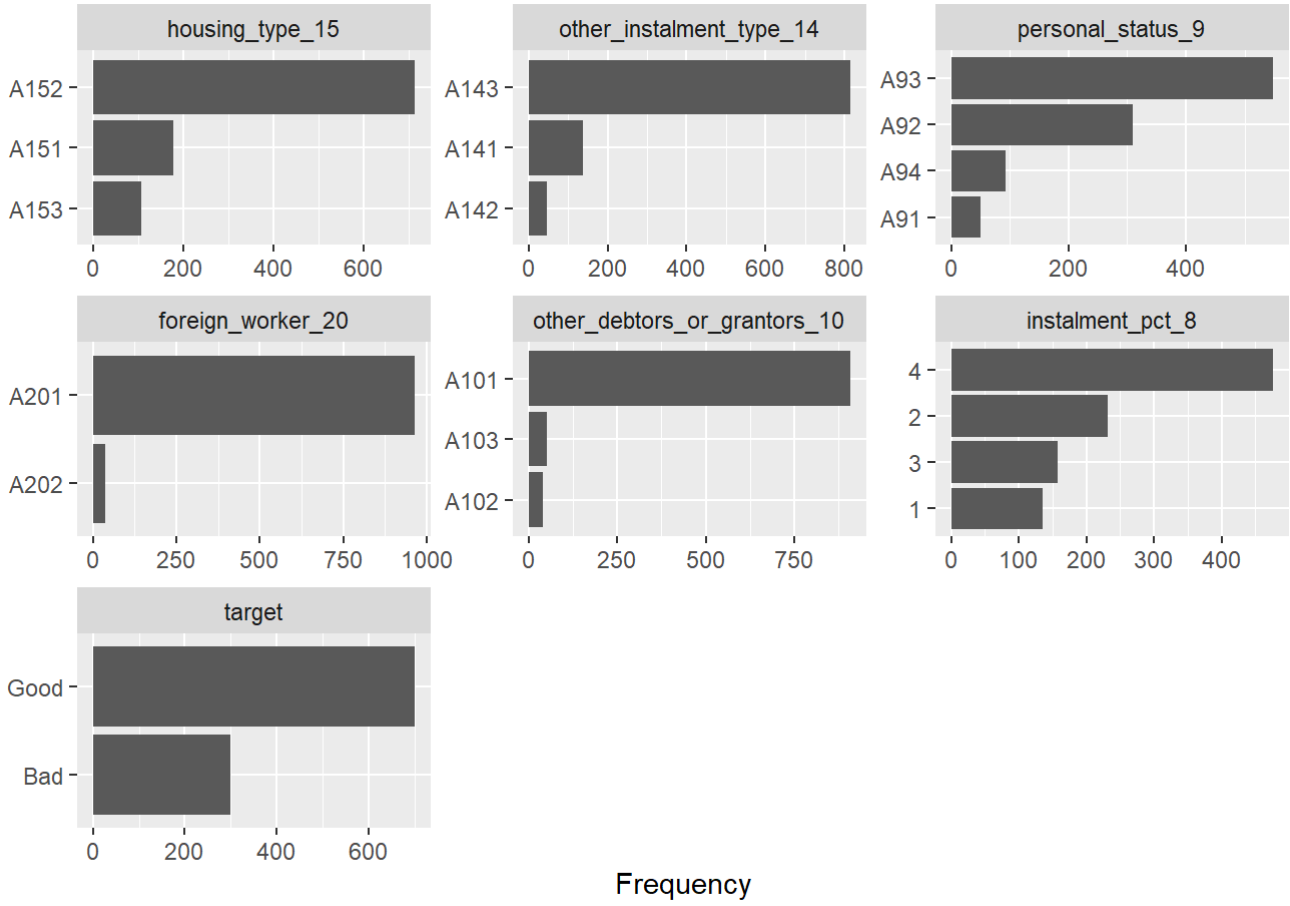
2.3. Análisis descriptivo (gráficos)

```
plot_intro(df) #gráfico para observar la distribución de variables y los casos missing por columnas, observaciones y filas
```




```
plot_bar(df) #gráfico para observar la distribución de frecuencias en variables categóricas
```





De las gráficas anteriores se puede observar:

1. La distribución de la target es adecuada y no necesita trabajo posterior.

2. Se puede observar que varias variables tienen algunas categorías con poca frecuencia. Sería oportuno analizar la conveniencia de recodificar en categorías con mayor representación.

3. Preparación para modelización

Particiones de training (70%) y test (30%)

Se segmenta la muestra en dos partes (train y test) empleando el programa Caret.

1. Training o entrenamiento (70% de la muestra): servirá para entrenar al modelo de clasificación.
2. Test (30%): servirá para validar el modelo. La característica fundamental es que esta muestra no debe haber tenido contacto previamente con el funcionamiento del modelo.

```
set.seed(100)  # Para reproducir los mismos resultados
partition <- createDataPartition(y = df$target, p = 0.7, list = FALSE)
train <- df[partition,]
test <- df[-partition,]
```

```
#Distribución de la variable TARGET
table(train$target)
```

```
##
##  Bad Good
##  210  490
```

```
table(test$target)
```

```
##
##  Bad Good
##   90  210
```

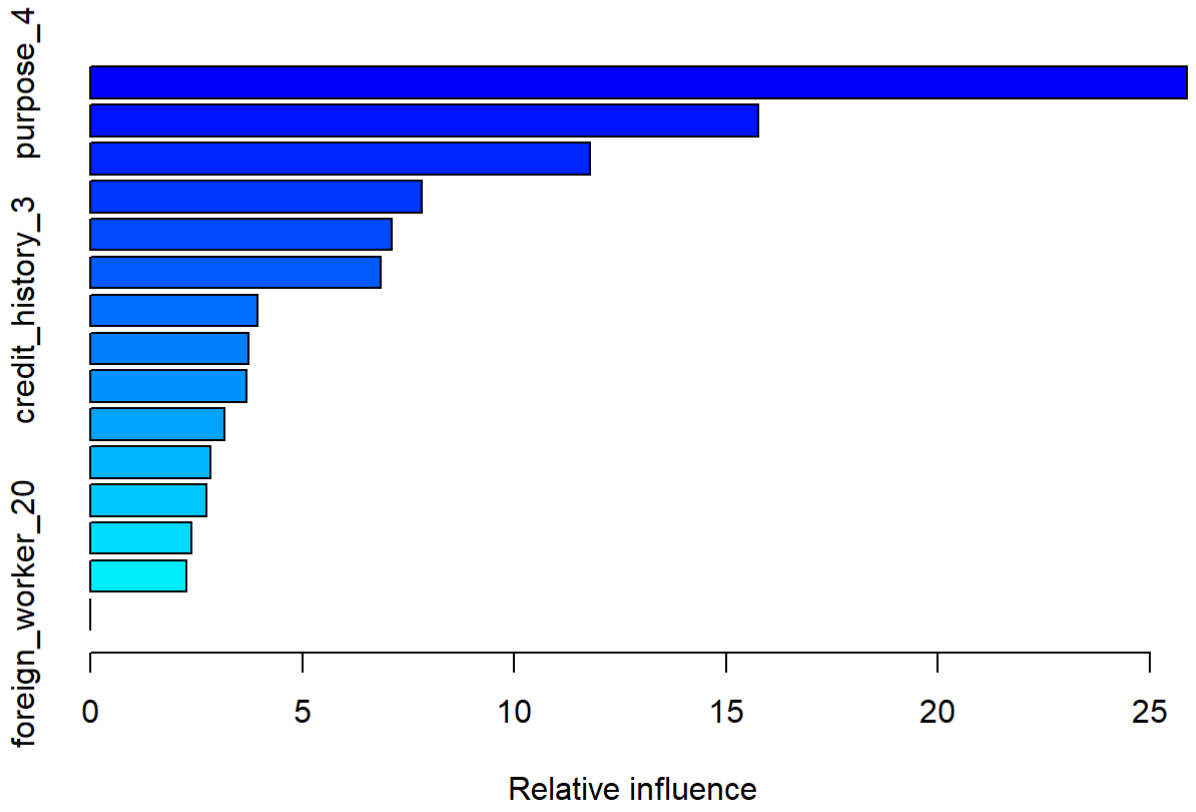
4. Modelización con Gradient Boosting Machine ("GBM Model")

Paso 1. Primer modelo

```
#library(gbm)
#Para casos de clasificación binaria resulta necesario recodificar la variable TARGET
 a numérica con valores 0 y 1.

#Además, para este tipo de casos de clasificación binaria se debe especificar la distr
ibution = "bernoulli".

# Se convierte TARGET 1 (Good) y 0 (Bad).
train$target <- ifelse(train$target == "Good", 1, 0)

# Entrenamiento con 5000 árboles.
set.seed(1)
gbm <- gbm(formula = target ~ .,
             distribution = "bernoulli",
             data = train,
             n.trees = 5000)

print(gbm)
```

```
## gbm(formula = target ~ ., distribution = "bernoulli", data = train,
##     n.trees = 5000)
## A gradient boosted model with bernoulli loss function.
## 5000 iterations were performed.
## There were 15 predictors of which 14 had non-zero influence.
```

```
summary(gbm)
```




```
##                                                    var   rel.inf
## purpose_4                                    purpose_4 25.869894
## age_in_yrs_13                            age_in_yrs_13 15.751720
## duration_month_2                      duration_month_2 11.785390
## credit_amount_5                        credit_amount_5  7.829651
## chk_ac_status_1                        chk_ac_status_1  7.117260
## p_employment_since_7              p_employment_since_7  6.859023
## credit_history_3                      credit_history_3  3.949338
## personal_status_9                    personal_status_9  3.730394
## property_type_12                      property_type_12  3.698943
## other_instalment_type_14        other_instalment_type_14  3.159761
## savings_ac_bond_6                      savings_ac_bond_6  2.843882
## instalment_pct_8                        instalment_pct_8  2.748018
## other_debtors_or_grantors_10 other_debtors_or_grantors_10  2.383513
## housing_type_15                          housing_type_15  2.273211
## foreign_worker_20                      foreign_worker_20  0.000000
```

En el gráfico anterior se observa la importancia relativa de las variables. Se opta por eliminar de los siguientes análisis la variable foreign_worker_20.

```
#Eliminamos de los df train y test la variable foreign_worker_20.
train <- select(train, -foreign_worker_20)
test <- select(test, -foreign_worker_20)
```

Paso 2. Segundo modelo

Aplicamos el segundo modelo en train

```
set.seed(1)
gbm1 <- gbm(formula = target ~ .,
             distribution = "bernoulli",
             data = train,
             n.trees = 5000)

print(gbm1)
```

```
## gbm(formula = target ~ ., distribution = "bernoulli", data = train,
##     n.trees = 5000)
## A gradient boosted model with bernoulli loss function.
## 5000 iterations were performed.
## There were 14 predictors of which 14 had non-zero influence.
```

Paso 3. Predict

```
# Se convierte TARGET en la muestra TEST a 1 y 0.
# Se emplea type = "response" para convertir los valores predichos en probabilidades.

test$target <- ifelse(test$target == "Good", 1, 0)

gbm_score <- predict(object = gbm1,
                     newdata = test,
                     n.trees = 5000,
                     type = "response")

# Observamos los seis primeros casos
head(gbm_score)
```

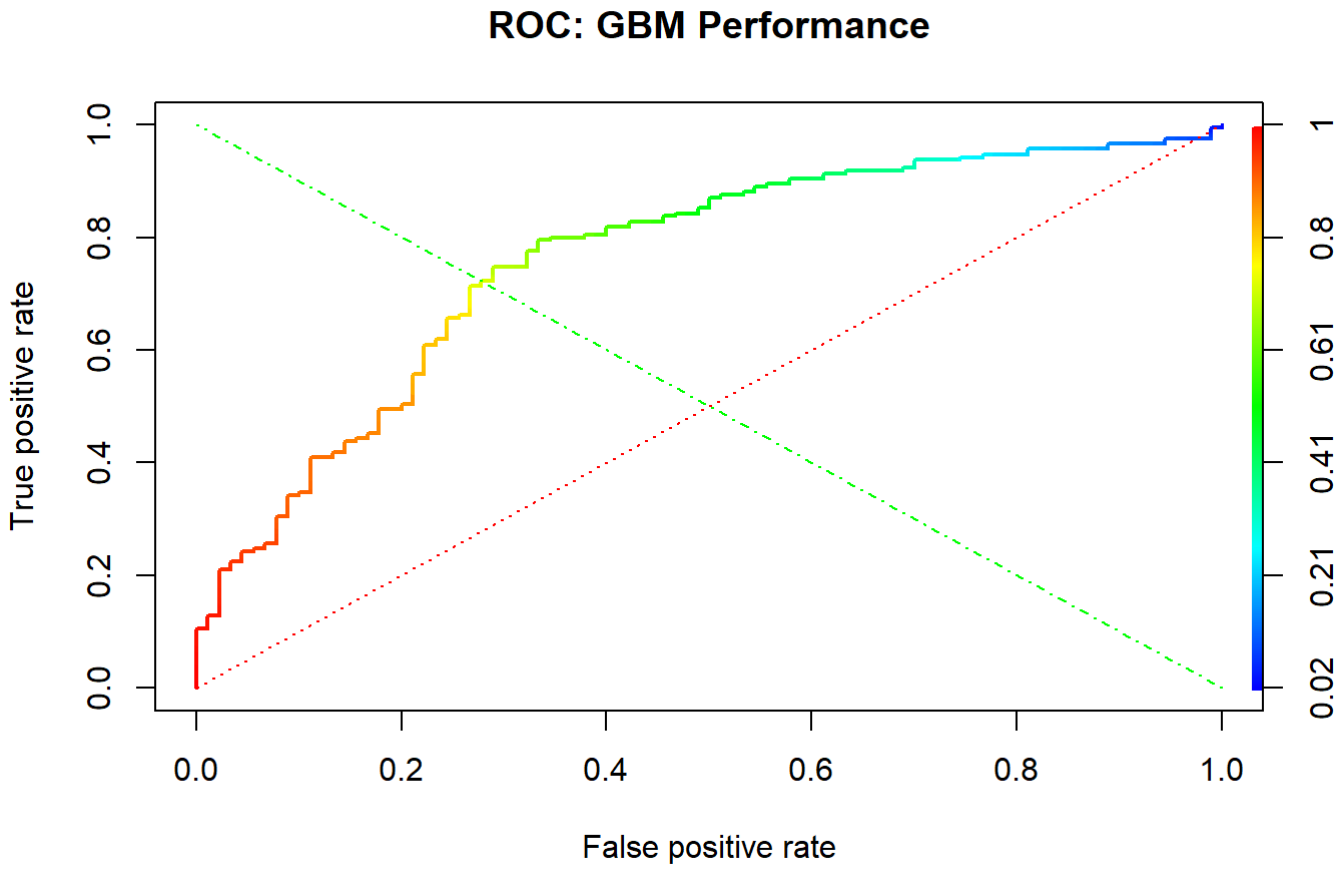
```
## [1] 0.9339782 0.9004792 0.9962055 0.7096389 0.2293012 0.7604590
```

Lanzamos un “head” para ver los 6 primeros. Lo que quiere decir que: el sujeto 1 tendrá una probabilidad de clasificarse como 1 (Goog credit) del 93,4%. El segundo de 90,05%, etc.

Paso 4. Curva ROC

Obtenemos la curva ROC

```
pred_gbm <- prediction(gbm_score, test$target)
perf_gbm <- performance(pred_gbm,"tpr","fpr")
#library(ROCR)
plot(perf_gbm, lwd=2, colorize=TRUE, main="ROC: GBM Performance")
lines(x=c(0, 1), y=c(0, 1), col="red", lwd=1, lty=3);
lines(x=c(1, 0), y=c(0, 1), col="green", lwd=1, lty=4)
```



Paso 5. Umbrales y matriz de confusión

A continuación se probarán distintos umbrales para maximizar el F1 al transformar la probabilidad obtenida en otra dicotómica (Good y Bad credit).

En otros proyectos hemos empleado funciones. En este caso lo haremos una por una para entender mejor le proceso. Lo que vamos cambiando es el umbral ("treshold"), observando en cada caso cómo varían las mátricas de la matriz de confusión (exactitud, sensibilidad, precisión y F1).

```
score2 <- ifelse(gbm_score > 0.20, "Good", "Bad")
MC <- table(test$target, score2)
Acc2 <- (MC[1,1] + MC[2,2]) / sum(MC) *100
Sen2 <- MC[2,2] / (MC[2,2] + MC[1,2]) *100
Pr2 <- MC[2,2] / (MC[2,2] + MC[2,1]) *100
F12 <- 2*Pr2*Sen2/(Pr2+Sen2)

score3 <- ifelse(gbm_score > 0.30, "Good", "Bad")
MC <- table(test$target, score3)
Acc3 <- (MC[1,1] + MC[2,2]) / sum(MC) *100
Sen3 <- MC[2,2] / (MC[2,2] + MC[1,2]) *100
Pr3 <- MC[2,2] / (MC[2,2] + MC[2,1]) *100
F13 <- 2*Pr3*Sen3/(Pr3+Sen3)

score4 <- ifelse(gbm_score > 0.40, "Good", "Bad")
MC <- table(test$target, score4)
Acc4 <- (MC[1,1] + MC[2,2]) / sum(MC) *100
Sen4 <- MC[2,2] / (MC[2,2] + MC[1,2]) *100
Pr4 <- MC[2,2] / (MC[2,2] + MC[2,1]) *100
F14 <- 2*Pr4*Sen4/(Pr4+Sen4)

score5 <- ifelse(gbm_score > 0.50, "Good", "Bad")
MC <- table(test$target, score5)
Acc5 <- (MC[1,1] + MC[2,2]) / sum(MC) *100
Sen5 <- MC[2,2] / (MC[2,2] + MC[1,2]) *100
Pr5 <- MC[2,2] / (MC[2,2] + MC[2,1]) *100
F15 <- 2*Pr5*Sen5/(Pr5+Sen5)

score6 <- ifelse(gbm_score > 0.60, "Good", "Bad")
MC <- table(test$target, score6)
Acc6 <- (MC[1,1] + MC[2,2]) / sum(MC) *100
Sen6 <- MC[2,2] / (MC[2,2] + MC[1,2]) *100
Pr6 <- MC[2,2] / (MC[2,2] + MC[2,1]) *100
F16 <- 2*Pr6*Sen6/(Pr6+Sen6)


#salida<-c(Acc2,Acc3,Acc4,Acc5,Acc6)
#salida
#salida<-c(Sen2,Sen3,Sen4,Sen5,Sen6)
#salida
#salida<-c(Pr2,Pr3,Pr4,Pr5,Pr6)
#salida
salida<-c(F12,F13,F14,F15,F16)
salida
```

```
## [1] 82.71605 83.29810 83.66013 82.13457 81.84019
```

Se puede observar que el límite donde se mazimiza la F1 es en 0,4, con un F1 = 83.66013

Paso 6. Métricas definitivas

```
#Matriz de confusión con umbral final
score4 <- ifelse(gbm_score > 0.40, "Good", "Bad")
MC <- table(test$target, score4)
gbm1_Acc <- round((MC[1,1] + MC[2,2]) / sum(MC) *100, 2)
gbm1_Sen <- round(MC[2,2] / (MC[2,2] + MC[1,2]) *100, 2)
gbm1_Pr <- round(MC[2,2] / (MC[2,2] + MC[2,1]) *100, 2)
gbm1_F1 <- round(2*gbm1_Pr*gbm1_Sen/(gbm1_Pr+gbm1_Sen), 2)

#KS & AUC
gbm1_KS <- round(max(attr(perf_gbm,'y.values')[[1]]-attr(perf_gbm,'x.values')[[1]])*10
0, 2)
gbm1_AUROC <- round(performance(pred_gbm, measure = "auc")@y.values[[1]]*100, 2)

#Métricas finales del modelo
cat("Acierto_gbm: ", gbm1_Acc,"\tSensibilidad_gbm: ", gbm1_Sen, "\tPrecision_gbm:", gb
m1_Pr, "\tF1-gbm:", gbm1_F1, "\tAUROC_gbm: ",gbm1_AUROC,"\tKS_gbm: ", gbm1_KS, "\n")
```

```
## Acierto_gbm:  75     Sensibilidad_gbm:  77.11    Precision_gbm: 91.43    F1-gbm: 8
3.66   AUROC_gbm:  76.06   KS_gbm:  46.19
```

Se obtiene una AUC de 76,06, lo que indica un modelo moderadamente aceptable.

5. Modelización con GBM y método OOB (tuning model)

Ajuste del modelo y "early stopping" en GBM

Usaremos la función **gbm.perf()** para estimar el número óptimo de iteraciones (n.trees) para un modelo de GM empleando dos métodos OOB (out-of-bag estimation) y CV (v-fold cross validation). Para una explicación más desarrollada, mirar en CRAN (https://cran.r-project.org/web/packages/gbm/vignettes/gbm.pdf)

Ambos son métodos de validación para realizar un "early stopping", esto es, determinar un número de árboles óptimo, en lugar del número más amplio con el que se comienza el proceso.

Paso 1. Primer modelo

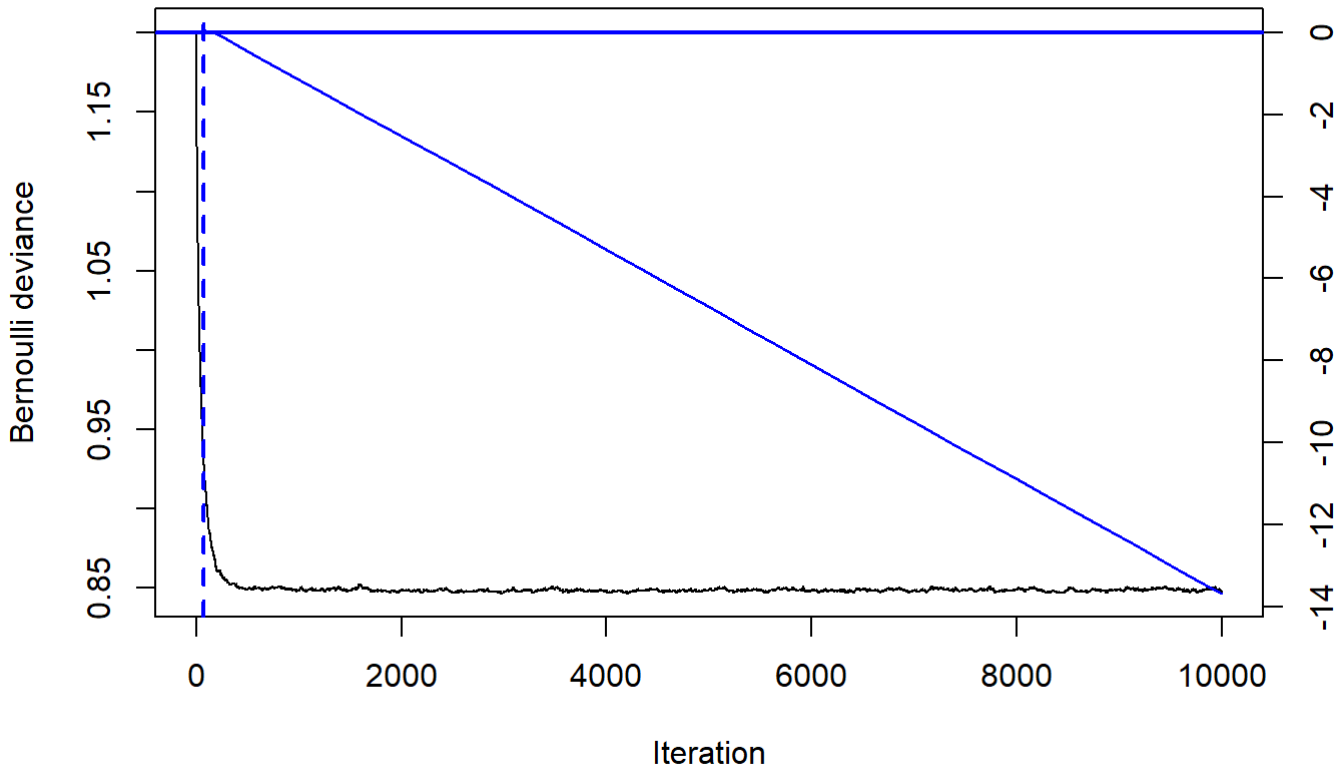
```
# Se entrena el modelo con 2 muestras de validación cruzada (cv.folds = 2)
set.seed(1)
gbm2 <- gbm(target ~ .,
                      distribution = "bernoulli",
                      data = train,
                      n.trees = 10000,
                      cv.folds = 2,
                      n.cores = 1)
```

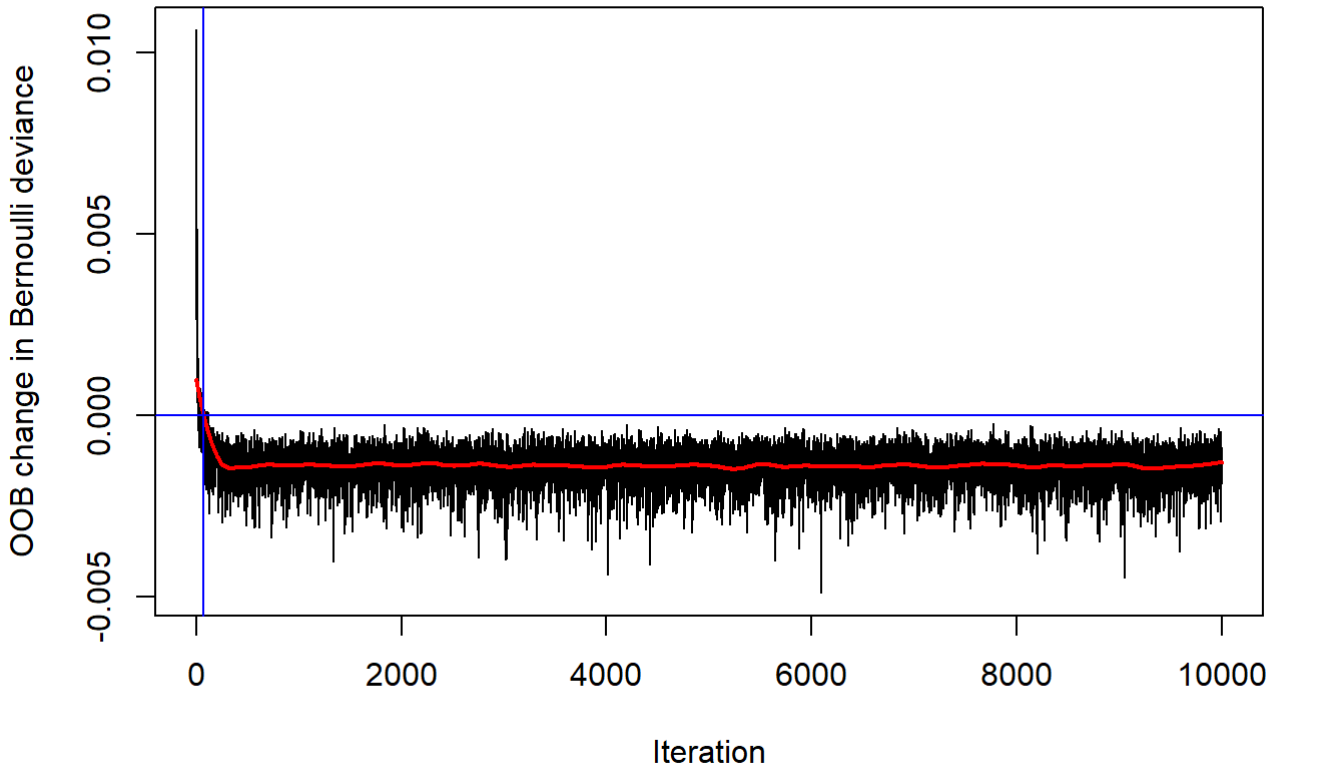
```
## CV: 1
## CV: 2
```

Paso 2. Número de áboles óptimo

```
ntree_opt_oob <- gbm.perf(object = gbm2,
                          method = "OOB",
                          oobag.curve = TRUE)
```

```
## OOB generally underestimates the optimal number of iterations although predictive p
erformance is reasonably competitive. Using cv_folds>1 when calling gbm usually result
s in improved predictive performance.
```






```
print(paste0("Optimal n.trees (OOB Estimate): ", ntree_opt_oob))
```

```
## [1] "Optimal n.trees (OOB Estimate): 73"
```

El número óptimo de árboles de 73

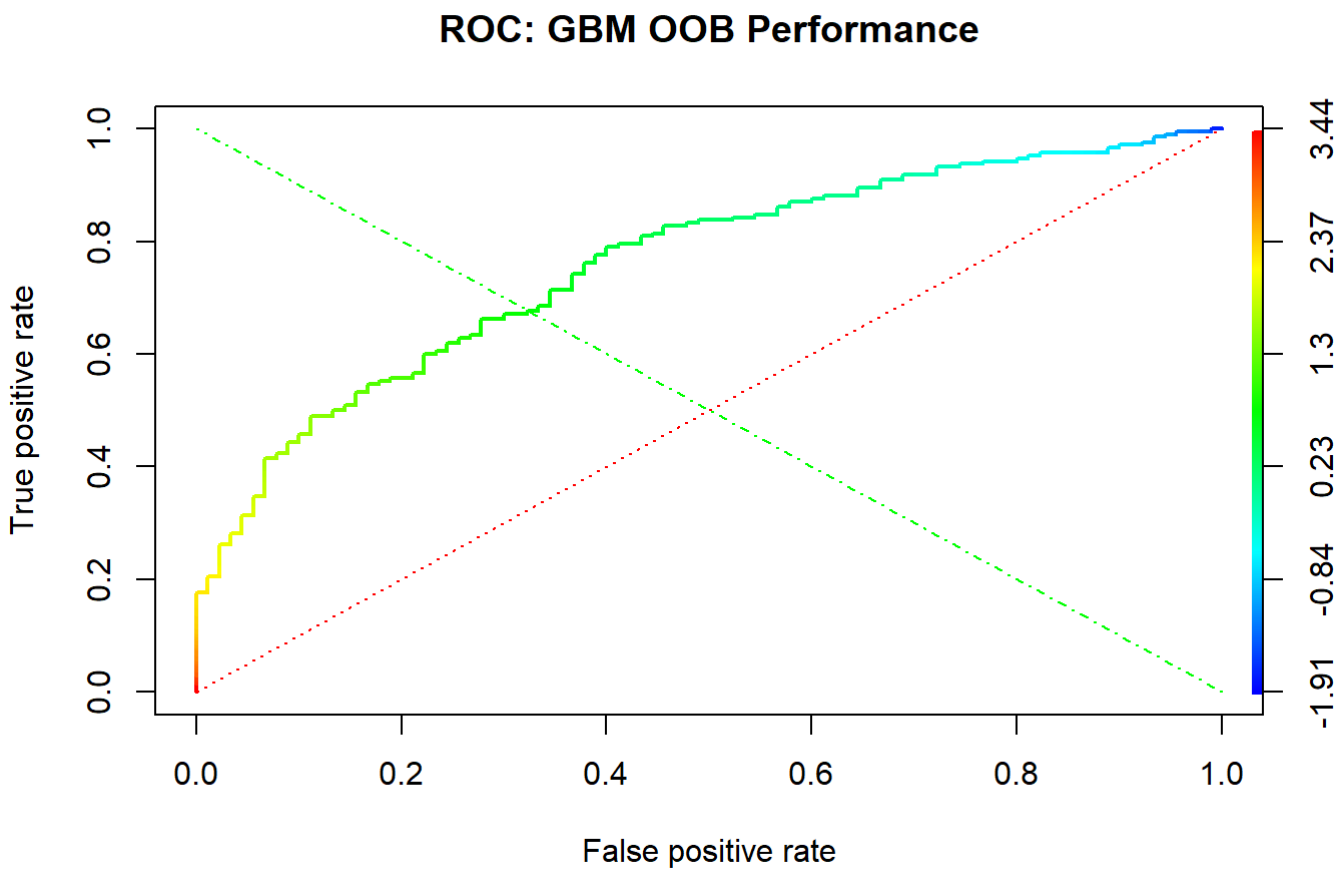
Paso 3. Predict

```
# Se obtienen las predicciones en la muestra de test.
gbm_score_oob <- predict(object = gbm2,
                     newdata = test,
                     n.trees = ntree_opt_oob)

head(gbm_score_oob)
```

```
## [1]  0.6265315  2.0290026  3.1451801  0.7988579 -0.0515592  0.7362966
```

Paso 4. Curva ROC

```
pred_gbm_oob <- prediction(gbm_score_oob, test$target)
perf_gbm_oob <- performance(pred_gbm_oob,"tpr","fpr")
#Library(ROCR)
plot(perf_gbm_oob, lwd=2, colorize=TRUE, main="ROC: GBM OOB Performance")
lines(x=c(0, 1), y=c(0, 1), col="red", lwd=1, lty=3);
lines(x=c(1, 0), y=c(0, 1), col="green", lwd=1, lty=4)
```



Paso 5. Umbrales y matriz de confusión

A continuación se probarán distintos umbrales para maximizar el F1 al transformar la probabilidad obtenida en otra dicotómica (Good y Bad credit).

En otros proyectos hemos empleado funciones. En este caso lo haremos una por una para entender mejor le proceso. Lo que vamos cambiando es el umbral (“treshold”), observando en cada caso cómo varían las mátricas de la matriz de confusión (exactitud, sensibilidad, precisión y F1).

```
score2 <- ifelse(gbm_score_oob > 0.20, "Good", "Bad")
MC <- table(test$target, score2)
Acc2 <- (MC[1,1] + MC[2,2]) / sum(MC) *100
Sen2 <- MC[2,2] / (MC[2,2] + MC[1,2]) *100
Pr2 <- MC[2,2] / (MC[2,2] + MC[2,1]) *100
F12 <- 2*Pr2*Sen2/(Pr2+Sen2)

score3 <- ifelse(gbm_score_oob > 0.30, "Good", "Bad")
MC <- table(test$target, score3)
Acc3 <- (MC[1,1] + MC[2,2]) / sum(MC) *100
Sen3 <- MC[2,2] / (MC[2,2] + MC[1,2]) *100
Pr3 <- MC[2,2] / (MC[2,2] + MC[2,1]) *100
F13 <- 2*Pr3*Sen3/(Pr3+Sen3)

score4 <- ifelse(gbm_score_oob > 0.40, "Good", "Bad")
MC <- table(test$target, score4)
Acc4 <- (MC[1,1] + MC[2,2]) / sum(MC) *100
Sen4 <- MC[2,2] / (MC[2,2] + MC[1,2]) *100
Pr4 <- MC[2,2] / (MC[2,2] + MC[2,1]) *100
F14 <- 2*Pr4*Sen4/(Pr4+Sen4)

score5 <- ifelse(gbm_score_oob > 0.50, "Good", "Bad")
MC <- table(test$target, score5)
Acc5 <- (MC[1,1] + MC[2,2]) / sum(MC) *100
Sen5 <- MC[2,2] / (MC[2,2] + MC[1,2]) *100
Pr5 <- MC[2,2] / (MC[2,2] + MC[2,1]) *100
F15 <- 2*Pr5*Sen5/(Pr5+Sen5)

score6 <- ifelse(gbm_score_oob > 0.60, "Good", "Bad")
MC <- table(test$target, score6)
Acc6 <- (MC[1,1] + MC[2,2]) / sum(MC) *100
Sen6 <- MC[2,2] / (MC[2,2] + MC[1,2]) *100
Pr6 <- MC[2,2] / (MC[2,2] + MC[2,1]) *100
F16 <- 2*Pr6*Sen6/(Pr6+Sen6)

#salida<-c(Acc2,Acc3,Acc4,Acc5,Acc6)
#salida
#salida<-c(Sen2,Sen3,Sen4,Sen5,Sen6)
#salida
#salida<-c(Pr2,Pr3,Pr4,Pr5,Pr6)
#salida
salida<-c(F12,F13,F14,F15,F16)
salida
```

```
## [1] 81.29330 81.77570 81.23515 79.60688 78.19549
```

Se puede observar que el límite donde se mazimiza la F1 es en 0,3, con un F1 = 81.77570

Paso 6. Métricas definitivas

```
#Matriz de confusión con umbral final
score3 <- ifelse(gbm_score_oob > 0.30, "Good", "Bad")
MC <- table(test$target, score3)
gbm_oob_Acc <- round((MC[1,1] + MC[2,2]) / sum(MC) *100, 2)
gbm_oob_Sen <- round(MC[2,2] / (MC[2,2] + MC[1,2]) *100, 2)
gbm_oob_Pr <- round(MC[2,2] / (MC[2,2] + MC[2,1]) *100, 2)
gbm_oob_F1 <- round(2*gbm_oob_Pr*gbm_oob_Sen/(gbm_oob_Pr+gbm_oob_Sen), 2)

#KS & AUC
gbm_oob_KS <- round(max(attr(perf_gbm_oob,'y.values')[[1]]-attr(perf_gbm_oob,'x.value
s')[[1]])*100, 2)
gbm_oob_AUROC <- round(performance(pred_gbm_oob, measure = "auc")@y.values[[1]]*100, 2
)

#Métricas finales del modelo
cat("Acierto_gbm_oob: ", gbm_oob_Acc,"\tSensibilidad_gbm_oob: ", gbm_oob_Sen, "\tPreci
sion_gbm_oob:", gbm_oob_Pr, "\tF1_gbm_oob:", gbm_oob_F1, "\tAUROC_gbm_oob: ",gbm_oob_A
UROC,"\tKS_gbm_oob: ", gbm_oob_KS, "\n")
```

```
## Acierto_gbm_oob:  74     Sensibilidad_gbm_oob:  80.28    Precision_gbm_oob: 83.33
F1_gbm_oob: 81.78   AUROC_gbm_oob:  75.99   KS_gbm_oob:  39.05
```

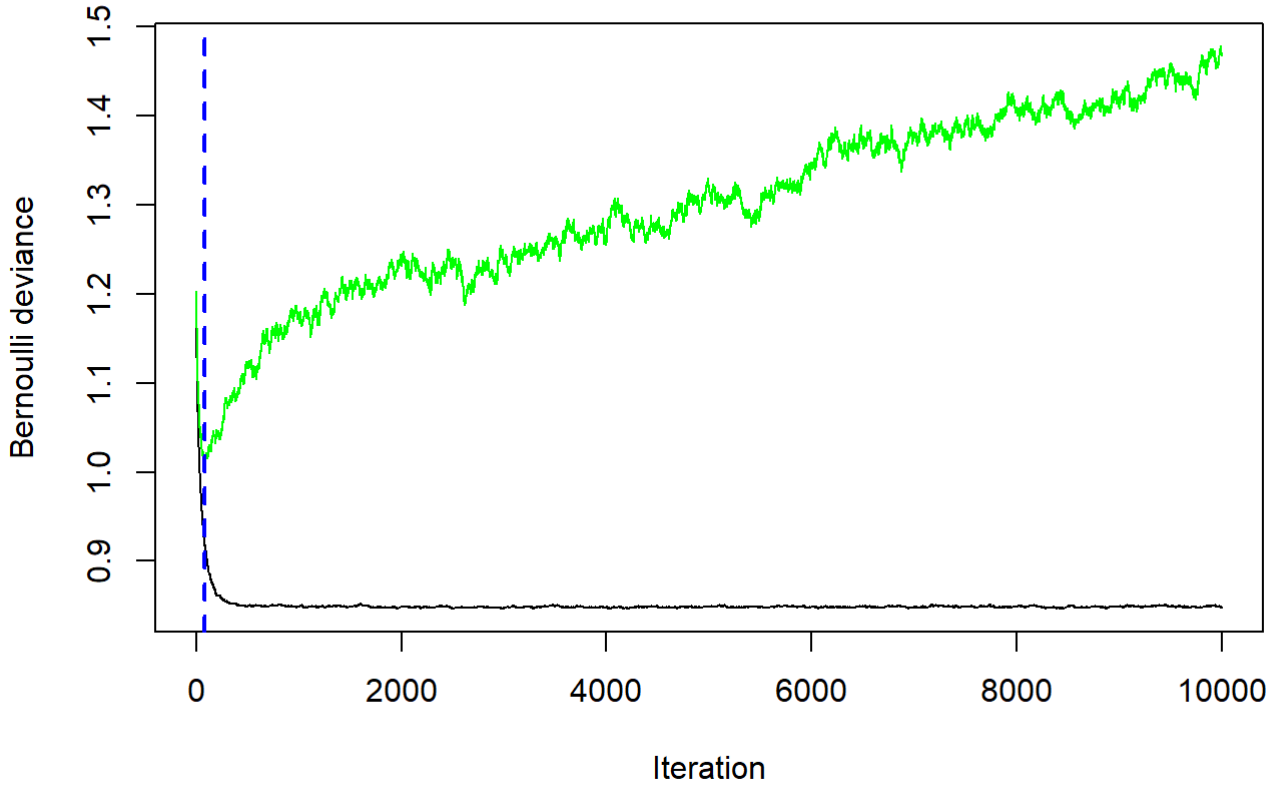
Empleando el método de validación OOB el AUC es de 75.99

6. Modelización con GBM y método CV (tuning model)

Manenemos el modelo gbm2.

Paso 2. Número de áboles óptimo

```
ntree_opt_cv <- gbm.perf(object = gbm2,
                         method = "cv")
```




```
print(paste0("Optimal n.trees (CV Estimate): ", ntree_opt_cv))
```

```
## [1] "Optimal n.trees (CV Estimate): 80"
```

El número de árboles óptimo es de 80.

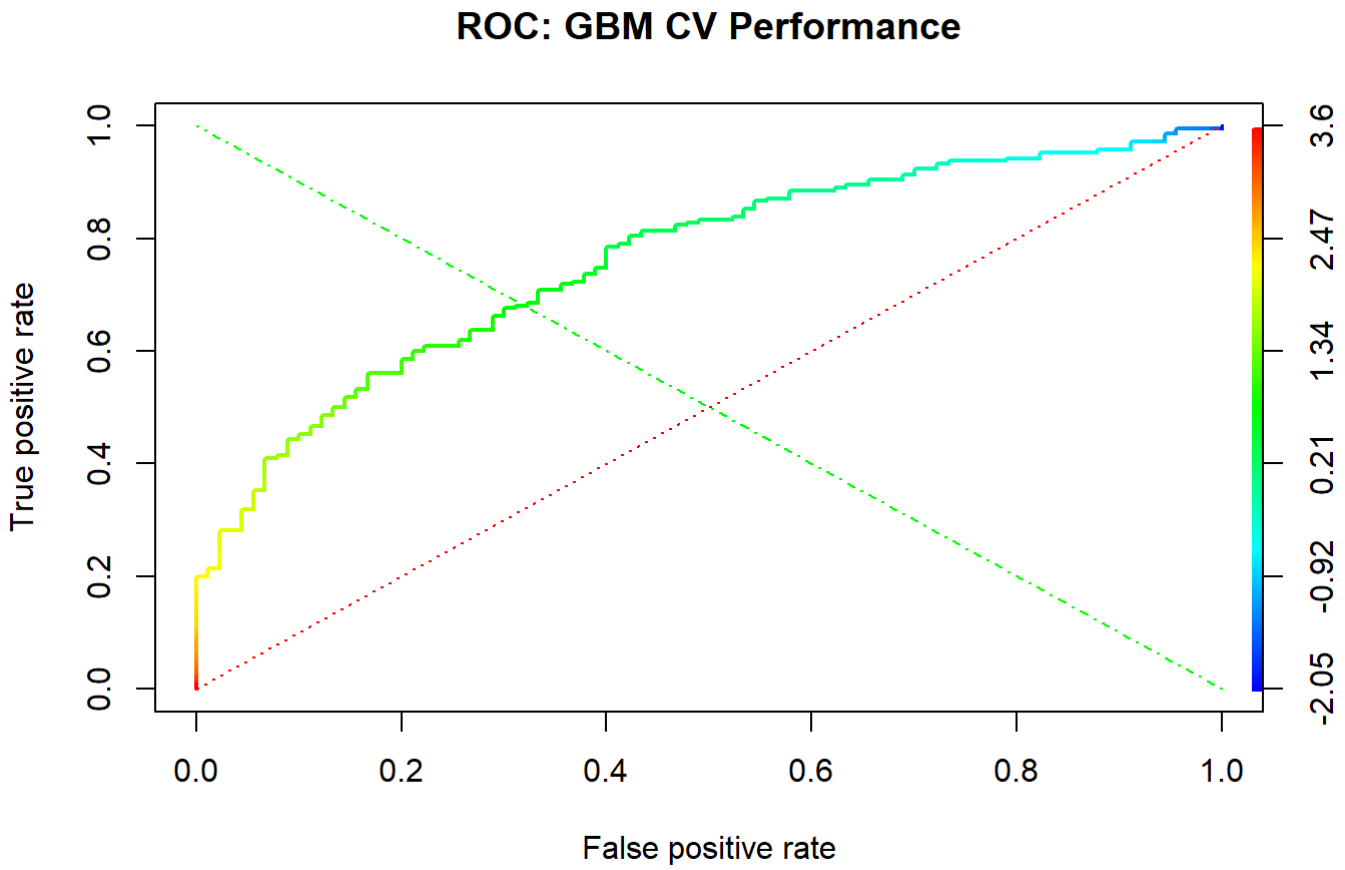
Paso 3. Predict

```
# Generate predictions on the test set using ntree_opt_cv number of trees
gbm_score_cv <- predict(object = gbm2,
                    newdata = test,
                    n.trees = ntree_opt_cv)

head(gbm_score_cv)
```

```
## [1]  0.66234733  2.05885791  3.24120506  0.82936682 -0.07978235  0.76467483
```

Paso 4. Curva ROC

```
pred_gbm_cv <- prediction(gbm_score_cv, test$target)
perf_gbm_cv <- performance(pred_gbm_cv,"tpr","fpr")
#Library(ROCR)
plot(perf_gbm_cv, lwd=2, colorize=TRUE, main="ROC: GBM CV Performance")
lines(x=c(0, 1), y=c(0, 1), col="red", lwd=1, lty=3);
lines(x=c(1, 0), y=c(0, 1), col="green", lwd=1, lty=4)
```



Paso 5. Umbrales y matriz de confusión

A continuación se probarán distintos umbrales para maximizar el F1 al transformar la probabilidad obtenida en otra dicotómica (Good y Bad credit).

En otros proyectos hemos empleado funciones. En este caso lo haremos una por una para entender mejor le proceso. Lo que vamos cambiando es el umbral ("treshold"), observando en cada caso cómo varían las mátricas de la matriz de confusión (exactitud, sensibilidad, precisión y F1).

```
score2 <- ifelse(gbm_score_cv > 0.20, "Good", "Bad")
MC <- table(test$target, score2)
Acc2 <- (MC[1,1] + MC[2,2]) / sum(MC) *100
Sen2 <- MC[2,2] / (MC[2,2] + MC[1,2]) *100
Pr2 <- MC[2,2] / (MC[2,2] + MC[2,1]) *100
F12 <- 2*Pr2*Sen2/(Pr2+Sen2)

score3 <- ifelse(gbm_score_cv > 0.30, "Good", "Bad")
MC <- table(test$target, score3)
Acc3 <- (MC[1,1] + MC[2,2]) / sum(MC) *100
Sen3 <- MC[2,2] / (MC[2,2] + MC[1,2]) *100
Pr3 <- MC[2,2] / (MC[2,2] + MC[2,1]) *100
F13 <- 2*Pr3*Sen3/(Pr3+Sen3)

score4 <- ifelse(gbm_score_cv > 0.40, "Good", "Bad")
MC <- table(test$target, score4)
Acc4 <- (MC[1,1] + MC[2,2]) / sum(MC) *100
Sen4 <- MC[2,2] / (MC[2,2] + MC[1,2]) *100
Pr4 <- MC[2,2] / (MC[2,2] + MC[2,1]) *100
F14 <- 2*Pr4*Sen4/(Pr4+Sen4)

score5 <- ifelse(gbm_score_cv > 0.50, "Good", "Bad")
MC <- table(test$target, score5)
Acc5 <- (MC[1,1] + MC[2,2]) / sum(MC) *100
Sen5 <- MC[2,2] / (MC[2,2] + MC[1,2]) *100
Pr5 <- MC[2,2] / (MC[2,2] + MC[2,1]) *100
F15 <- 2*Pr5*Sen5/(Pr5+Sen5)

score6 <- ifelse(gbm_score_cv > 0.60, "Good", "Bad")
MC <- table(test$target, score6)
Acc6 <- (MC[1,1] + MC[2,2]) / sum(MC) *100
Sen6 <- MC[2,2] / (MC[2,2] + MC[1,2]) *100
Pr6 <- MC[2,2] / (MC[2,2] + MC[2,1]) *100
F16 <- 2*Pr6*Sen6/(Pr6+Sen6)


#salida<-c(Acc2,Acc3,Acc4,Acc5,Acc6)
#salida
#salida<-c(Sen2,Sen3,Sen4,Sen5,Sen6)
#salida
#salida<-c(Pr2,Pr3,Pr4,Pr5,Pr6)
#salida
salida<-c(F12,F13,F14,F15,F16)
salida
```

```
## [1] 81.01852 81.49883 80.48193 79.11548 77.38693
```

Se puede observar que el límite donde se mazimiza la F1 es en 0,3, con un F1 = 81.49883

Paso 6. Métricas definitivas

```
#Matriz de confusión con umbral final
score3 <- ifelse(gbm_score_cv > 0.30, "Good", "Bad")
MC <- table(test$target, score3)
gbm_cv_Acc <- round((MC[1,1] + MC[2,2]) / sum(MC) *100, 2)
gbm_cv_Sen <- round(MC[2,2] / (MC[2,2] + MC[1,2]) *100, 2)
gbm_cv_Pr <- round(MC[2,2] / (MC[2,2] + MC[2,1]) *100, 2)
gbm_cv_F1 <- round(2*gbm_cv_Pr*gbm_cv_Sen/(gbm_cv_Pr+gbm_cv_Sen), 2)

#KS & AUC
gbm_cv_KS <- round(max(attr(perf_gbm_cv,'y.values')[[1]]-attr(perf_gbm_cv,'x.values')
[[1]])*100, 2)
gbm_cv_AUROC <- round(performance(pred_gbm_cv, measure = "auc")@y.values[[1]]*100, 2)

#Métricas finales del modelo
cat("Acierto_gbm_cv: ", gbm_cv_Acc,"\tSensibilidad_gbm_cv: ", gbm_cv_Sen, "\tPrecision
_gbm_cv:", gbm_cv_Pr, "\tF1-gbm_cv:", gbm_cv_F1, "\tAUROC_gbm_cv: ",gbm_cv_AUROC,"\tKS
_gbm_cv: ", gbm_cv_KS, "\n")
```

```
## Acierto_gbm_cv:  73.67   Sensibilidad_gbm_cv:  80.18     Precision_gbm_cv: 82.86
F1-gbm_cv: 81.5     AUROC_gbm_cv:  76.07    KS_gbm_cv:  39.52
```

Empleando el método de validación OOB el AUC es de 76.07

7. Comparación de los tres modelos

```
# Etiquetas de filas
models <- c('GBM', 'GBM oob', 'GBM cv')

#Accuracy
models_Acc <- c(gbm1_Acc, gbm_oob_Acc, gbm_cv_Acc)

#Sensibilidad
models_Sen <- c(gbm1_Sen, gbm_oob_Sen, gbm_cv_Sen)

#Precisión
models_Pr <- c(gbm1_Pr, gbm_oob_Pr, gbm_cv_Pr)

#F1
models_F1 <- c(gbm1_F1, gbm_oob_F1, gbm_cv_F1)

# AUC
models_AUC <- c(gbm1_AUROC, gbm_oob_AUROC, gbm_cv_AUROC)

# KS
models_KS <- c(gbm1_KS, gbm_oob_KS, gbm_cv_KS)
```

```
# Combinar métricas
metricas <- as.data.frame(cbind(models, models_Acc, models_Sen, models_Pr, models_F1,
 models_AUC, models_KS))
```

```
# Colnames
colnames(metricas) <- c("Model", "Acc", "Sen", "Pr", "F1", "AUC", "KS")
```

```
# Tabla final de métricas
kable(metricas, caption ="Comparision of Model Performances")
```

Comparision of Model Performances

Model	Acc	Sen	Pr	F1	AUC	KS
GBM	75	77.11	91.43	83.66	76.06	46.19
GBM oob	74	80.28	83.33	81.78	75.99	39.05
GBM cv	73.67	80.18	82.86	81.5	76.07	39.52

Conclusión:

Se observa que los tres modelos son muy semejantes, siendo ligeramente superior en la AUC el modelo con el ajuste CV.