Gradient Boosting Machine ajustando todos los hiperparámetros

Adolfo Sánchez Burón

Algoritmos empleados: Gradient Boosting Machine (GBM)

Para una breve descripción del algoritmo GBM mirar el post (https://www.ml2projects.com/post/gbm-oov-cv)

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

En este post se va a seguir el proceso seguido por Boehmke y Hernández ajustando todos los hiperparámetros.

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

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

1. Entorno

1.1. Instalar librerías

```
library(dplyr)       # Manipulación de datos
library(knitr)       # Para formato de tablas
library(ROCR)        # Rendimiento del modelo y curva ROC
library(caret)       # División de muestra, Clasificación y regresión
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 ...
```

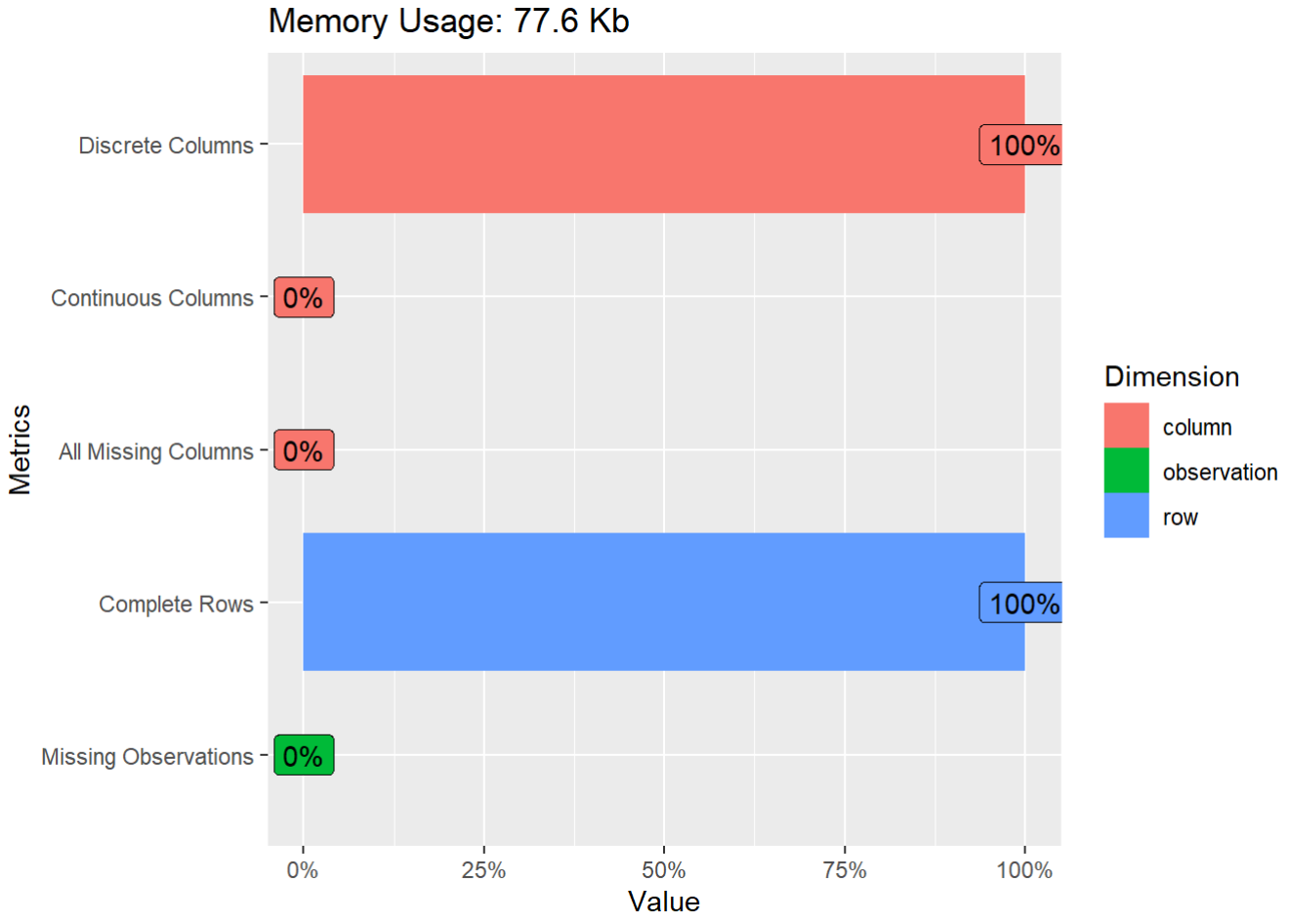
```
lapply(df,summary) #mostrar la distribución de frecuencias en cada categoría de todas
 las variables
```

```
## $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
```

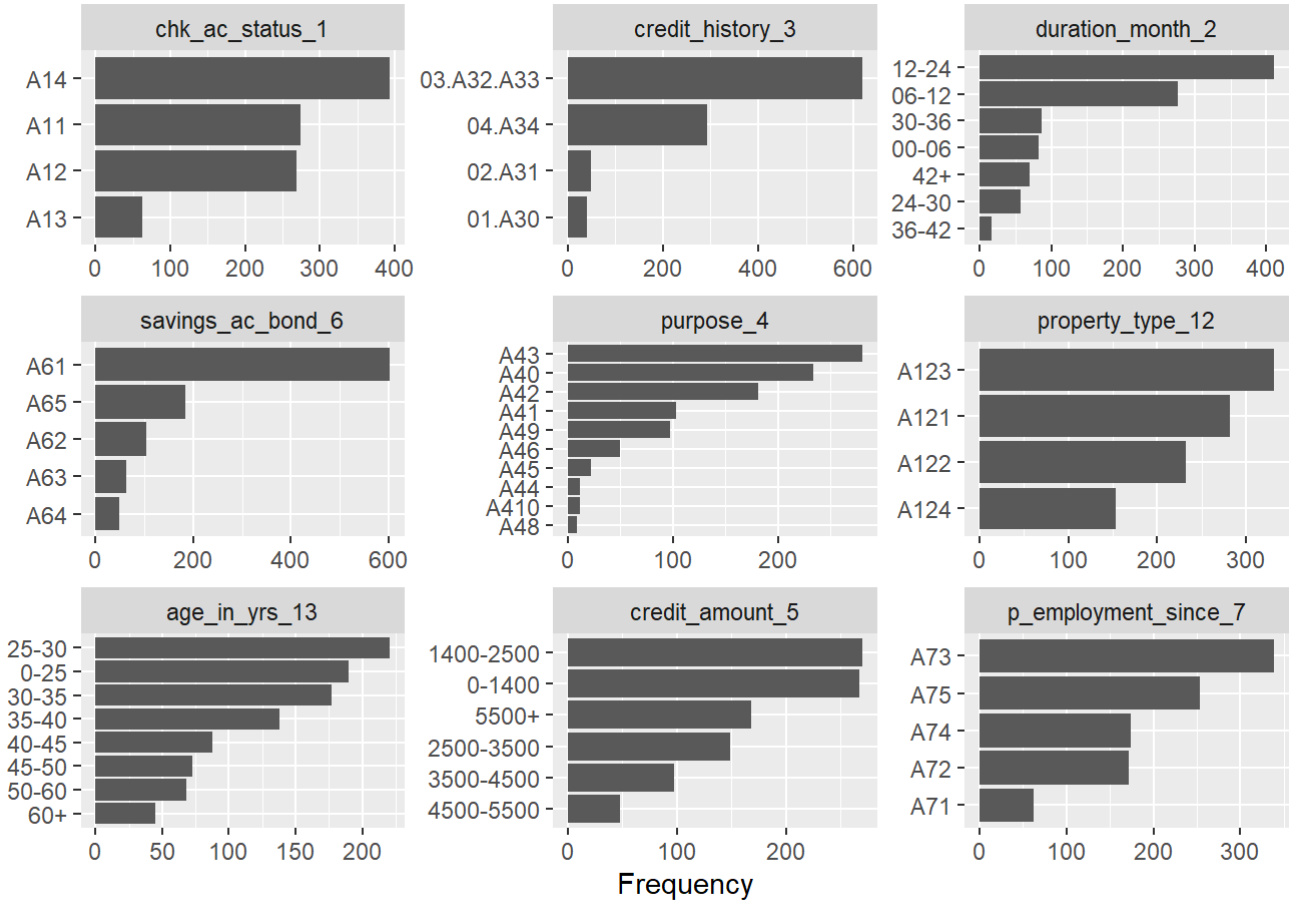
```
##
## $other_debtors_or_grantors_10
## A101 A102 A103
##  907   41   52
##
## $instalment_pct_8
##   1   2   3   4
## 136 231 157 476
##
## $target
##  Bad Good
##  300  700
```

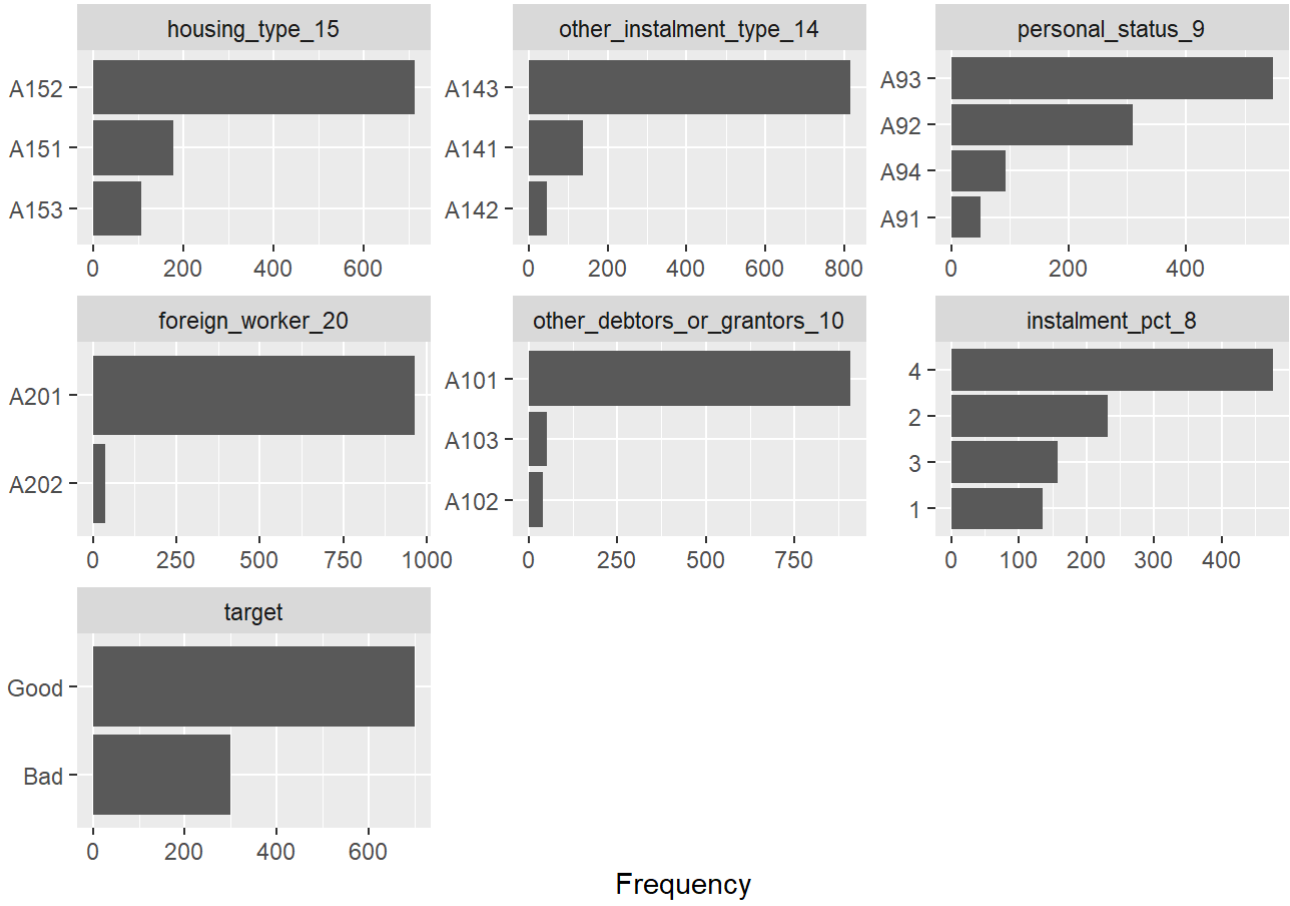
2.3. Análisis descriptivo (gráficos)

```
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,]
```

Comprobamos la distribución de la TARGET en las dos muestras.

```
table(train$target)
```

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

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

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

4. Modelización con GBM con parámetros ajustados (tuning model)

Paso 1. Primer modelo

Eliminamos la variable con nulo poder predictivo (observado en análisis previos).

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

```
#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)


set.seed(123)

# Entrenamos el modelo
gbm.fit <- gbm(
  formula = target ~ .,
  distribution = "bernoulli",
  data = train,
  n.trees = 5000,
  interaction.depth = 1,
  shrinkage = 0.001,
  cv.folds = 5,
  n.cores = NULL, # will use all cores by default
  verbose = FALSE
  )

# Imprimimos resultados
print(gbm.fit)
```

```
## gbm(formula = target ~ ., distribution = "bernoulli", data = train,
##     n.trees = 5000, interaction.depth = 1, shrinkage = 0.001,
##     cv.folds = 5, verbose = FALSE, n.cores = NULL)
## A gradient boosted model with bernoulli loss function.
## 5000 iterations were performed.
## The best cross-validation iteration was 5000.
## There were 14 predictors of which 14 had non-zero influence.
```

```
# Obtenemos el RMSE (root mean square error)
sqrt(min(gbm.fit$cv.error))
```

```
## [1] 1.014304
```

Paso 2. Extracción de prámetros óptimos

```
# Preparamos el ajuste de los hiperparámetros
hyper_grid <- expand.grid(
  shrinkage = c(0.01, 0.1, 0.3),
  interaction.depth = c(1, 3, 5),
  n.minobsinnode = c(5, 10, 15),
  bag.fraction = c(0.65, 0.8, 1),
  optimal_trees = 0,
  min_RMSE = 0,
  min_cor = 0
)

nrow(hyper_grid) # Número total de combinaciones
```

```
## [1] 81
```

```
# Para hacer una búsqueda automática de los huperparámetros, aleatorizamos los datos
random_index <- sample(1:nrow(train), nrow(train))
random_train <- train[random_index, ]

for(i in 1:nrow(hyper_grid)) {
  set.seed(123)
  gbm.tune <- gbm(
    formula = target ~ .,
    distribution = "bernoulli",
    data = train,
    n.trees = 5000,
    interaction.depth = hyper_grid$interaction.depth[i],
    shrinkage = hyper_grid$shrinkage[i],
    n.minobsinnode = hyper_grid$n.minobsinnode[i],
    bag.fraction = hyper_grid$bag.fraction[i],
    train.fraction = 0.75,
    n.cores = NULL,
    verbose = FALSE
  )

# Agregamos a hyper_grid la información que nos interesa que nos interesa
  hyper_grid$optimal_trees[i] <- which.min(gbm.tune$valid.error)
  hyper_grid$min_RMSE[i] <- sqrt(min(gbm.tune$valid.error))
  hyper_grid$min_cor[i] <- cor(random_train$target, predict(gbm.tune))
}
```

```
## Using 1163 trees...
```

```
## Using 116 trees...
```

```
## Using 25 trees...
```

```
## Using 587 trees...
```

```
## Using 67 trees...
```

```
## Using 19 trees...
```

```
## Using 355 trees...
```

```
## Using 27 trees...
```

```
## Using 8 trees...
```

```
## Using 1163 trees...
```

```
## Using 116 trees...
```

```
## Using 25 trees...
```

```
## Using 576 trees...
```

```
## Using 67 trees...
```

```
## Using 14 trees...
```

```
## Using 443 trees...
```

```
## Using 39 trees...
```

```
## Using 13 trees...
```

```
## Using 1163 trees...
```

```
## Using 116 trees...
```

```
## Using 27 trees...
```

```
## Using 576 trees...
```

```
## Using 51 trees...
```

```
## Using 20 trees...
```

```
## Using 446 trees...
```

```
## Using 25 trees...
```

```
## Using 13 trees...
```

```
## Using 1327 trees...
```

```
## Using 136 trees...
```

```
## Using 50 trees...
```

```
## Using 625 trees...
```

```
## Using 49 trees...
```

```
## Using 10 trees...
```

```
## Using 456 trees...
```

```
## Using 45 trees...
```

```
## Using 13 trees...
```

```
## Using 1327 trees...
```

```
## Using 156 trees...
```

```
## Using 50 trees...
```

```
## Using 588 trees...
```

```
## Using 57 trees...
```

```
## Using 10 trees...
```

```
## Using 400 trees...
```

```
## Using 45 trees...
```

```
## Using 18 trees...
```

```
## Using 1327 trees...
```

```
## Using 136 trees...
```

```
## Using 52 trees...
```

```
## Using 580 trees...
```

```
## Using 84 trees...
```

```
## Using 10 trees...
```

```
## Using 459 trees...
```

```
## Using 43 trees...
```

```
## Using 16 trees...
```

```
## Using 2156 trees...
```

```
## Using 230 trees...
```

```
## Using 54 trees...
```

```
## Using 448 trees...
```

```
## Using 61 trees...
```

```
## Using 15 trees...
```

```
## Using 322 trees...
```

```
## Using 60 trees...
```

```
## Using 9 trees...
```

```
## Using 2156 trees...
```

```
## Using 230 trees...
```

```
## Using 54 trees...
```

```
## Using 746 trees...
```

```
## Using 77 trees...
```

```
## Using 17 trees...
```

```
## Using 356 trees...
```

```
## Using 49 trees...
```

```
## Using 22 trees...
```

```
## Using 2156 trees...
```

```
## Using 230 trees...
```

```
## Using 54 trees...
```

```
## Using 798 trees...
```

```
## Using 78 trees...
```

```
## Using 12 trees...
```

```
## Using 409 trees...
```

```
## Using 41 trees...
```

```
## Using 11 trees...
```

Sacamos los resultados obtenidos en hyper_grid (shrinkage, interaction.depth, n.minobsinnode, bag.fraction, optimal_trees, min_RMSE, min_cor)

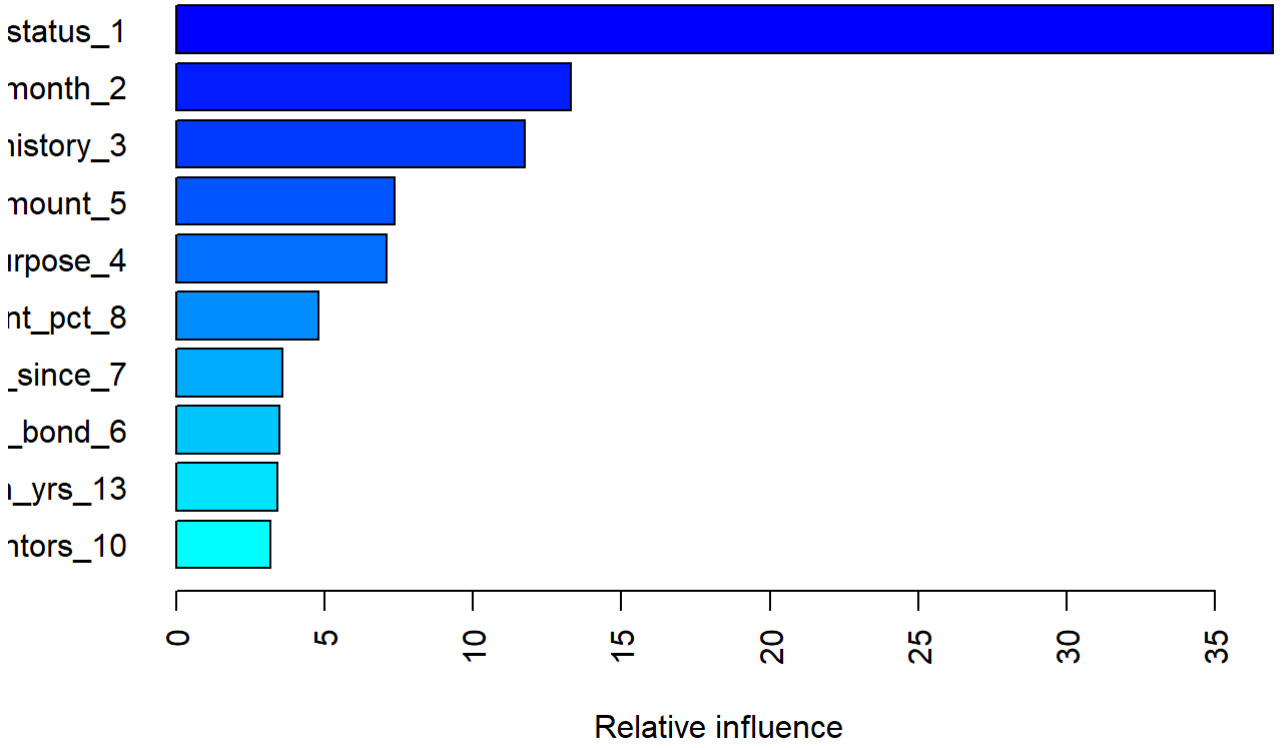
```
hyper_grid %>%
  dplyr::arrange(min_RMSE) %>%
  head(10)
```

```
##    shrinkage interaction.depth n.minobsinnode bag.fraction optimal_trees
## 1       0.30                 1             15         0.65            27
## 2       0.30                 1              5         0.65            25
## 3       0.30                 1             10         0.65            25
## 4       0.30                 1             15         0.80            52
## 5       0.10                 3             15         0.80            84
## 6       0.30                 1              5         0.80            50
## 7       0.30                 1             10         0.80            50
## 8       0.01                 3             15         0.65           576
## 9       0.10                 1             15         0.65           116
## 10      0.10                 1             10         0.65           116
##     min_RMSE       min_cor
## 1  0.9753025  0.0002388511
## 2  0.9781977  0.0054256153
## 3  0.9781977  0.0054256153
## 4  0.9788572  0.0027679504
## 5  0.9813145  0.0066744948
## 6  0.9835245 -0.0032494533
## 7  0.9835245 -0.0032494533
## 8  0.9835855  0.0001318378
## 9  0.9840560  0.0034281250
## 10 0.9842841  0.0004677813
```

Se observa en la tabla que el mínimo RMSE es: 0.9753025 (la RMSE del modelo anterior era 1.014). Pasamos estos valores al modelo definitivo.

Paso 3. Modelo definitivo con prámetros ajustados

```
set.seed(123)

# train GBM modelEntrenamiento del modelo
gbm.fit.final <- gbm(
  formula = target ~ .,
  distribution = "bernoulli",
  data = train,
  n.trees = 27,
  interaction.depth = 1,
  shrinkage = 0.3,
  n.minobsinnode = 15,
  bag.fraction = 0.65,
  train.fraction = 1,
  n.cores = NULL,
  verbose = FALSE
  )

summary(gbm.fit.final, cBars = 10,
        method = relative.influence, las = 2)
```

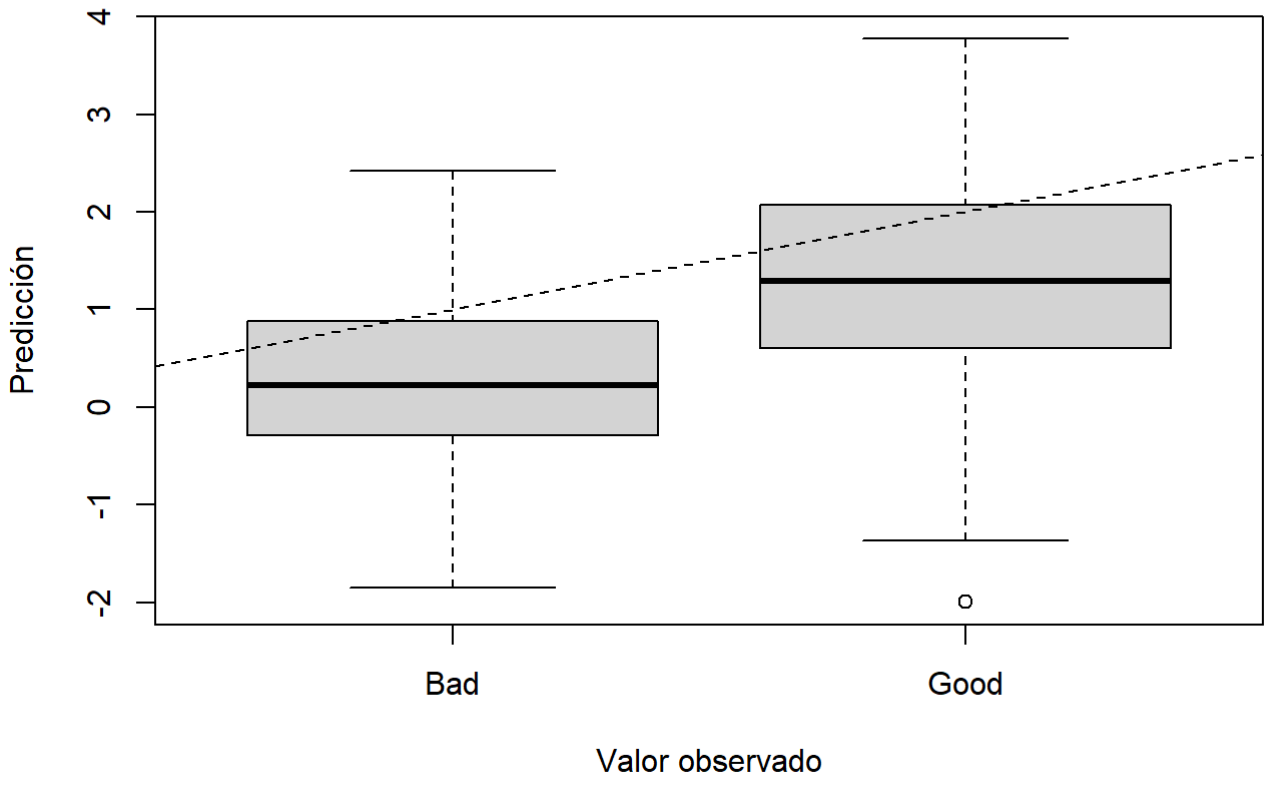


```
##                                                   var   rel.inf
## chk_ac_status_1                         chk_ac_status_1 36.951331
## duration_month_2                       duration_month_2 13.292565
## credit_history_3                       credit_history_3 11.758416
## credit_amount_5                         credit_amount_5  7.365959
## purpose_4                                     purpose_4  7.088495
## instalment_pct_8                       instalment_pct_8  4.785093
## p_employment_since_7                 p_employment_since_7  3.581935
## savings_ac_bond_6                       savings_ac_bond_6  3.476691
## age_in_yrs_13                             age_in_yrs_13  3.414112
## other_debtors_or_grantors_10 other_debtors_or_grantors_10  3.174170
## other_instalment_type_14         other_instalment_type_14  3.059566
## property_type_12                         property_type_12  2.051668
## housing_type_15                           housing_type_15  0.000000
## personal_status_9                       personal_status_9  0.000000
```

Podemos observar que hay dos variables con nulo poder predictivo en el modelo: housing_type_15 y personal_status_9. Posteriormente haremos otro modelo excluyendo estas dos variables.

Paso 4. Predict

```
gbm_pred <- predict(object=gbm.fit.final, newdata=test, n.trees = 27)

plot(x=test$target, y=gbm_pred, xlab='Valor observado', ylab='Predicción')
abline(a=0, b=1, lty='dashed')
```

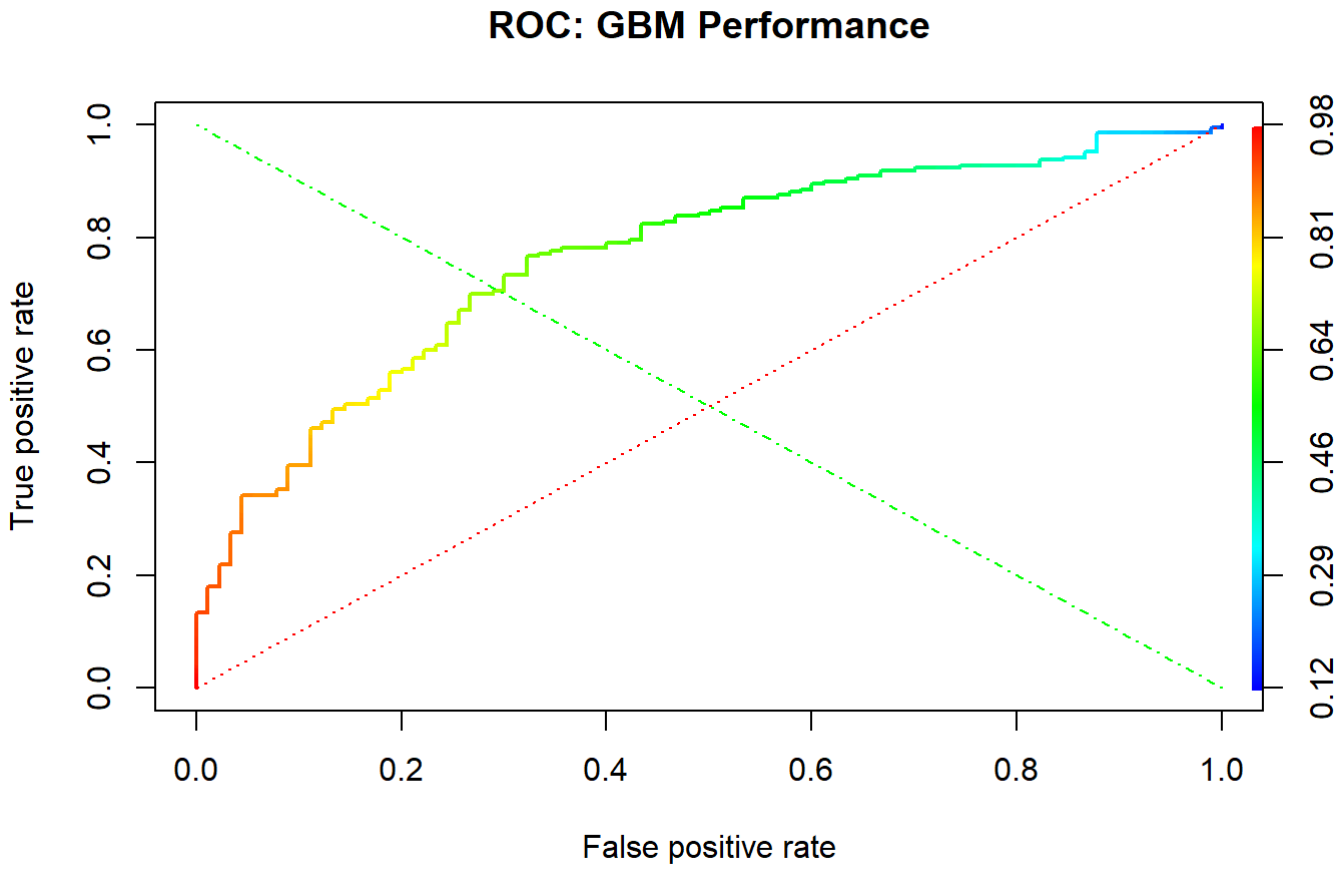


```
# Se convierte TARGET en la muestra TEST a 1 y 0.
#test$target <- ifelse(test$target == "Good", 1, 0)

# Se emplea type = "response" para convertir los valores predichos en probabilidades.

gbm_score <- predict(object = gbm.fit.final,
                     newdata = test,
                     n.trees = 27,
                     type = "response")

head(gbm_score)
```

```
## [1] 0.7426707 0.9086818 0.9561684 0.7117533 0.4612036 0.6618296
```

Paso 5. 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 6. 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] 82.28346 82.96593 81.66667 83.07692 80.57554
```

Se puede observar que el límite donde se mazimiza la F1 es en 0,5, con un F1 = 83.07692

Paso 7. Métricas definitivas

```
#Matriz de confusión con umbral final
score5 <- ifelse(gbm_score > 0.50, "Good", "Bad")
MC <- table(test$target, score5)
gbm_Acc_T1 <- round((MC[1,1] + MC[2,2]) / sum(MC) *100, 2)
gbm_Sen_T1 <- round(MC[2,2] / (MC[2,2] + MC[1,2]) *100, 2)
gbm_Pr_T1 <- round(MC[2,2] / (MC[2,2] + MC[2,1]) *100, 2)
gbm_F1_T1 <- round(2*gbm_Pr_T1*gbm_Sen_T1/(gbm_Pr_T1+gbm_Sen_T1), 2)

#KS & AUC
gbm_KS_T1 <- round(max(attr(perf_gbm,'y.values')[[1]]-attr(perf_gbm,'x.values')[[1]])*
100, 2)
gbm_AUROC_T1 <- round(performance(pred_gbm, measure = "auc")@y.values[[1]]*100, 2)

#Métricas finales del modelo
cat("Acierto_gbm: ", gbm_Acc_T1,"\tSensibilidad_gbm: ", gbm_Sen_T1, "\tPrecision_gbm:"
, gbm_Pr_T1, "\tF1-gbm:", gbm_F1_T1, "\tAUROC_gbm: ",gbm_AUROC_T1,"\tKS_gbm: ", gbm_KS
_T1, "\n")
```

```
## Acierto_gbm:  74.33  Sensibilidad_gbm:  77.14    Precision_gbm: 90   F1-gbm: 83.08
AUROC_gbm:  76.48   KS_gbm:  44.44
```

Se obtiene una AUC de 76,48, lo que indica un modelo moderadamente aceptable.

5. Modelización con GBM con parámetros ajustados (tuning model) con variables eliminadas

Paso 1. Primer modelo

```
#Eliminamos de los df train y test la variable foreign_worker_20.
train <- select(train, -housing_type_15, -personal_status_9)
test <- select(test, -housing_type_15, -personal_status_9)
```

```
set.seed(123)

# Entrenamos modelo
gbm2.fit <- gbm(
  formula = target ~ .,
  distribution = "bernoulli",
  data = train,
  n.trees = 5000,
  interaction.depth = 1,
  shrinkage = 0.001,
  cv.folds = 5,
  n.cores = NULL,
  verbose = FALSE
  )

# Imprimimos resultados
print(gbm2.fit)
```

```
## gbm(formula = target ~ ., distribution = "bernoulli", data = train,
##     n.trees = 5000, interaction.depth = 1, shrinkage = 0.001,
##     cv.folds = 5, verbose = FALSE, n.cores = NULL)
## A gradient boosted model with bernoulli loss function.
## 5000 iterations were performed.
## The best cross-validation iteration was 5000.
## There were 12 predictors of which 12 had non-zero influence.
```

Paso 2. Extracción de prámetros óptimos

```
# create hyperparameter grid
hyper_grid2 <- expand.grid(
  shrinkage = c(0.01, 0.1, 0.3),
  interaction.depth = c(1, 3, 5),
  n.minobsinnode = c(5, 10, 15),
  bag.fraction = c(0.65, 0.8, 1),
  optimal_trees = 0,               # a place to dump results
  min_RMSE = 0,                    # a place to dump results
  min_cor = 0
)

nrow(hyper_grid2) # total number of combinations
```

```
## [1] 81
```

```
# randomize data
random_index <- sample(1:nrow(train), nrow(train))
random_train <- train[random_index, ]

for(i in 1:nrow(hyper_grid2)) {
  set.seed(123)
  gbm.tune2 <- gbm(
    formula = target ~ .,
    distribution = "bernoulli",
    data = train,
    n.trees = 5000,
    interaction.depth = hyper_grid$interaction.depth[i],
    shrinkage = hyper_grid$shrinkage[i],
    n.minobsinnode = hyper_grid$n.minobsinnode[i],
    bag.fraction = hyper_grid$bag.fraction[i],
    train.fraction = 0.75,
    n.cores = NULL, # will use all cores by default
    verbose = FALSE
  )

# Agregamos la información que nos interesa
  hyper_grid2$optimal_trees[i] <- which.min(gbm.tune2$valid.error)
  hyper_grid2$min_RMSE[i] <- sqrt(min(gbm.tune2$valid.error))
  hyper_grid2$min_cor[i] <- cor(random_train$target, predict(gbm.tune2))
}
```

```
## Using 1481 trees...
```

```
## Using 175 trees...
```

```
## Using 50 trees...
```

```
## Using 659 trees...
```

```
## Using 63 trees...
```

```
## Using 19 trees...
```

```
## Using 406 trees...
```

```
## Using 63 trees...
```

```
## Using 13 trees...
```

```
## Using 1481 trees...
```

```
## Using 175 trees...
```

```
## Using 50 trees...
```

```
## Using 586 trees...
```

```
## Using 64 trees...
```

```
## Using 13 trees...
```

```
## Using 434 trees...
```

```
## Using 50 trees...
```

```
## Using 19 trees...
```

```
## Using 1487 trees...
```

```
## Using 123 trees...
```

```
## Using 50 trees...
```

```
## Using 659 trees...
```

```
## Using 51 trees...
```

```
## Using 19 trees...
```

```
## Using 441 trees...
```

```
## Using 50 trees...
```

```
## Using 13 trees...
```

```
## Using 1709 trees...
```

```
## Using 182 trees...
```

```
## Using 52 trees...
```

```
## Using 604 trees...
```

```
## Using 82 trees...
```

```
## Using 11 trees...
```

```
## Using 456 trees...
```

```
## Using 48 trees...
```

```
## Using 7 trees...
```

```
## Using 1709 trees...
```

```
## Using 168 trees...
```

```
## Using 52 trees...
```

```
## Using 703 trees...
```

```
## Using 49 trees...
```

```
## Using 13 trees...
```

```
## Using 470 trees...
```

```
## Using 42 trees...
```

```
## Using 10 trees...
```

```
## Using 1682 trees...
```

```
## Using 188 trees...
```

```
## Using 52 trees...
```

```
## Using 700 trees...
```

```
## Using 79 trees...
```

```
## Using 13 trees...
```

```
## Using 441 trees...
```

```
## Using 36 trees...
```

```
## Using 11 trees...
```

```
## Using 2363 trees...
```

```
## Using 225 trees...
```

```
## Using 56 trees...
```

```
## Using 618 trees...
```

```
## Using 66 trees...
```

```
## Using 11 trees...
```

```
## Using 406 trees...
```

```
## Using 48 trees...
```

```
## Using 8 trees...
```

```
## Using 2363 trees...
```

```
## Using 225 trees...
```

```
## Using 56 trees...
```

```
## Using 686 trees...
```

```
## Using 72 trees...
```

```
## Using 17 trees...
```

```
## Using 384 trees...
```

```
## Using 54 trees...
```

```
## Using 9 trees...
```

```
## Using 2440 trees...
```

```
## Using 225 trees...
```

```
## Using 56 trees...
```

```
## Using 721 trees...
```

```
## Using 63 trees...
```

```
## Using 25 trees...
```

```
## Using 406 trees...
```

```
## Using 42 trees...
```

```
## Using 17 trees...
```

Sacamos los resultados obtenidos en hyper_grid (shrinkage, interaction.depth, n.minobsinnode, bag.fraction, optimal_trees, min_RMSE, min_cor)

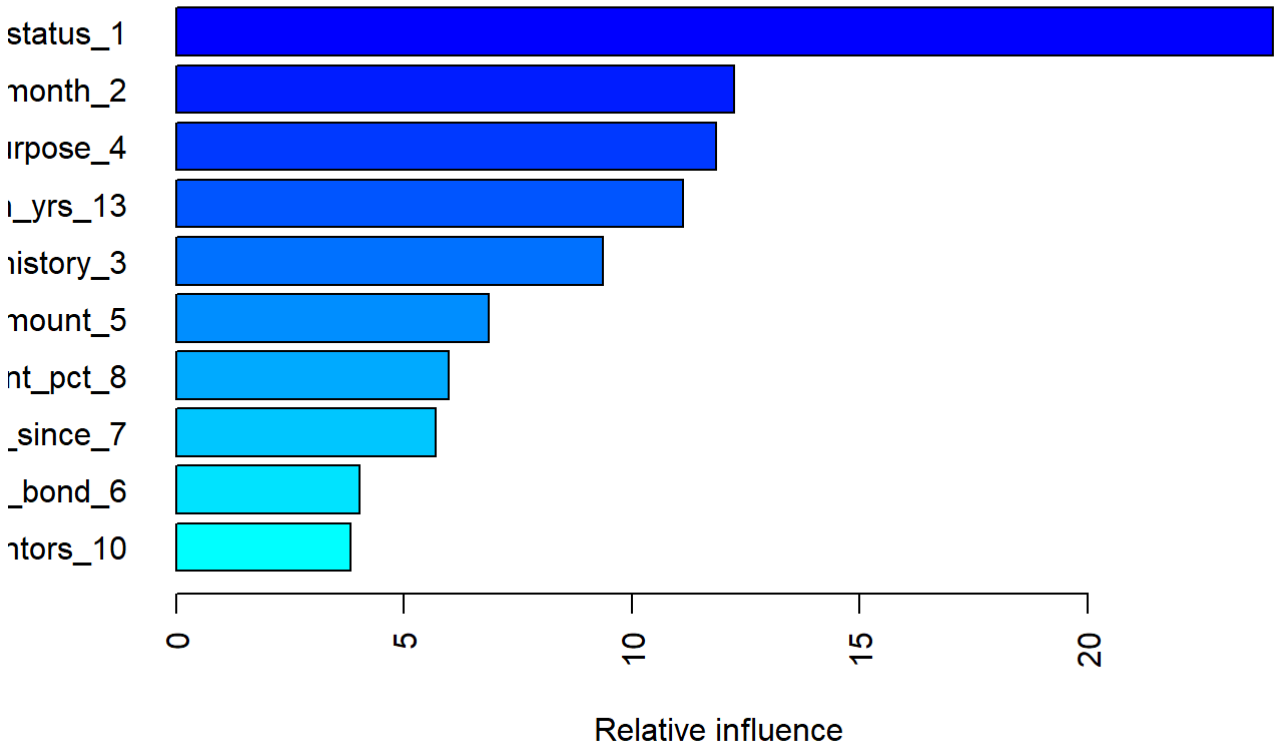
```
hyper_grid2 %>%
  dplyr::arrange(min_RMSE) %>%
  head(10)
```

```
##    shrinkage interaction.depth n.minobsinnode bag.fraction optimal_trees
## 1        0.3                 3              5         0.65            19
## 2        0.1                 5             10         0.65            50
## 3        0.3                 1              5         0.65            50
## 4        0.3                 1             10         0.65            50
## 5        0.3                 1             15         0.65            50
## 6        0.1                 3             15         0.80            79
## 7        0.3                 1             15         0.80            52
## 8        0.3                 1              5         0.80            52
## 9        0.3                 3             15         0.65            19
## 10       0.3                 1             10         0.80            52
##     min_RMSE       min_cor
## 1  0.9583456  0.0083676439
## 2  0.9666268 -0.0073234424
## 3  0.9690286  0.0141520221
## 4  0.9720435  0.0087717349
## 5  0.9740683  0.0047509046
## 6  0.9750708  0.0060058506
## 7  0.9753483 -0.0034874517
## 8  0.9755129  0.0026841467
## 9  0.9760739 -0.0061937110
## 10 0.9761165  0.0004921278
```

Se observa en la tabla que el mínimo RMSE es: 0.9583456 (la RMSE del modelo anterior era 0.9753025). Pasamos estos valores al modelo definitivo.

Paso 3. Modelo definitivo con prámetros ajustados

```
set.seed(123)

# Entrenamos el modelo
gbm.fit.final2 <- gbm(
  formula = target ~ .,
  distribution = "bernoulli",
  data = train,
  n.trees = 19,
  interaction.depth = 3,
  shrinkage = 0.3,
  n.minobsinnode = 5,
  bag.fraction = 0.65,
  train.fraction = 1,
  n.cores = NULL,
  verbose = FALSE
  )

summary(gbm.fit.final2, cBars = 10,
        method = relative.influence, las = 2)
```

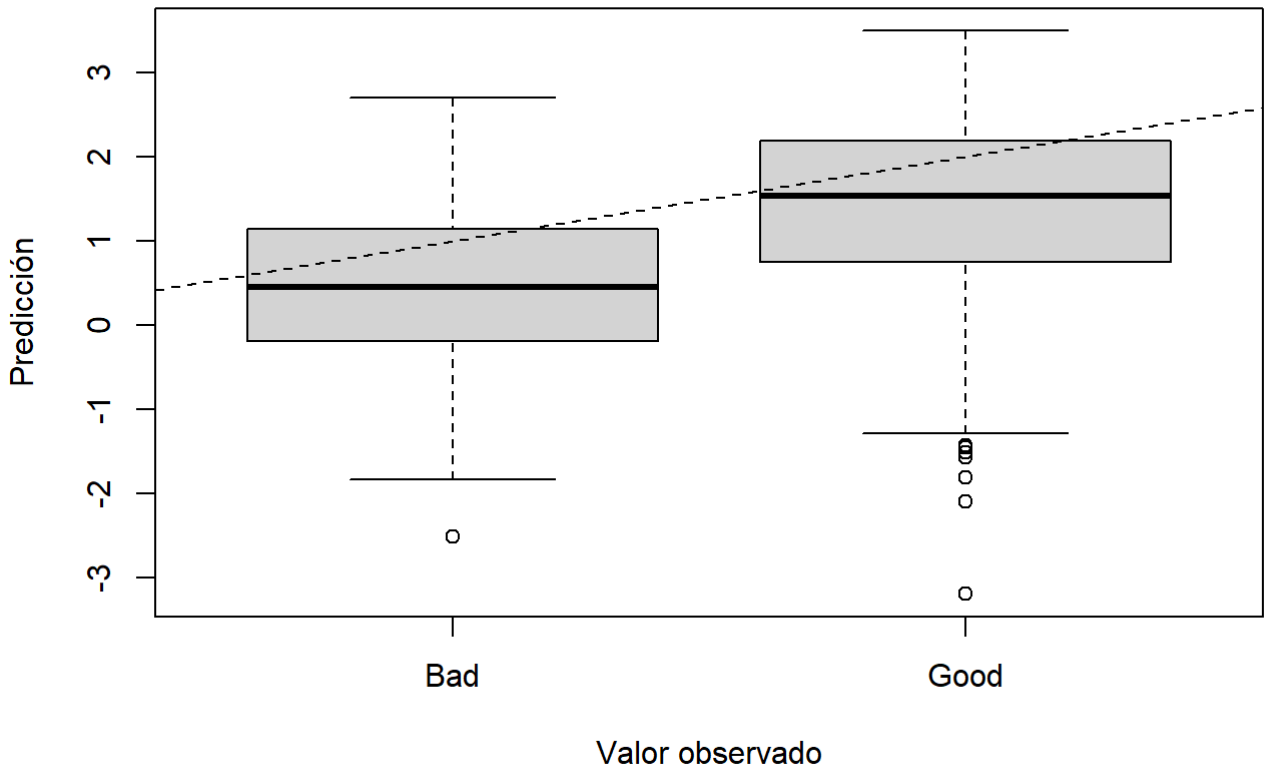



```
##                                                    var   rel.inf
## chk_ac_status_1                          chk_ac_status_1 24.073744
## duration_month_2                        duration_month_2 12.246941
## purpose_4                                      purpose_4 11.855482
## age_in_yrs_13                             age_in_yrs_13 11.118319
## credit_history_3                        credit_history_3  9.371801
## credit_amount_5                          credit_amount_5  6.857989
## instalment_pct_8                        instalment_pct_8  5.984810
## p_employment_since_7                p_employment_since_7  5.686218
## savings_ac_bond_6                      savings_ac_bond_6  4.027205
## other_debtors_or_grantors_10 other_debtors_or_grantors_10  3.827067
## other_instalment_type_14         other_instalment_type_14  2.521020
## property_type_12                         property_type_12  2.429402
```

Ahora vemos que no hay ninguna variables con peso predictivo nulo.

Paso 4. Predict

```
gbm_pred2 <- predict(object=gbm.fit.final2, newdata=test, n.trees = 19)

plot(x=test$target, y=gbm_pred2, xlab='Valor observado', ylab='Predicción')
abline(a=0, b=1, lty='dashed')
```

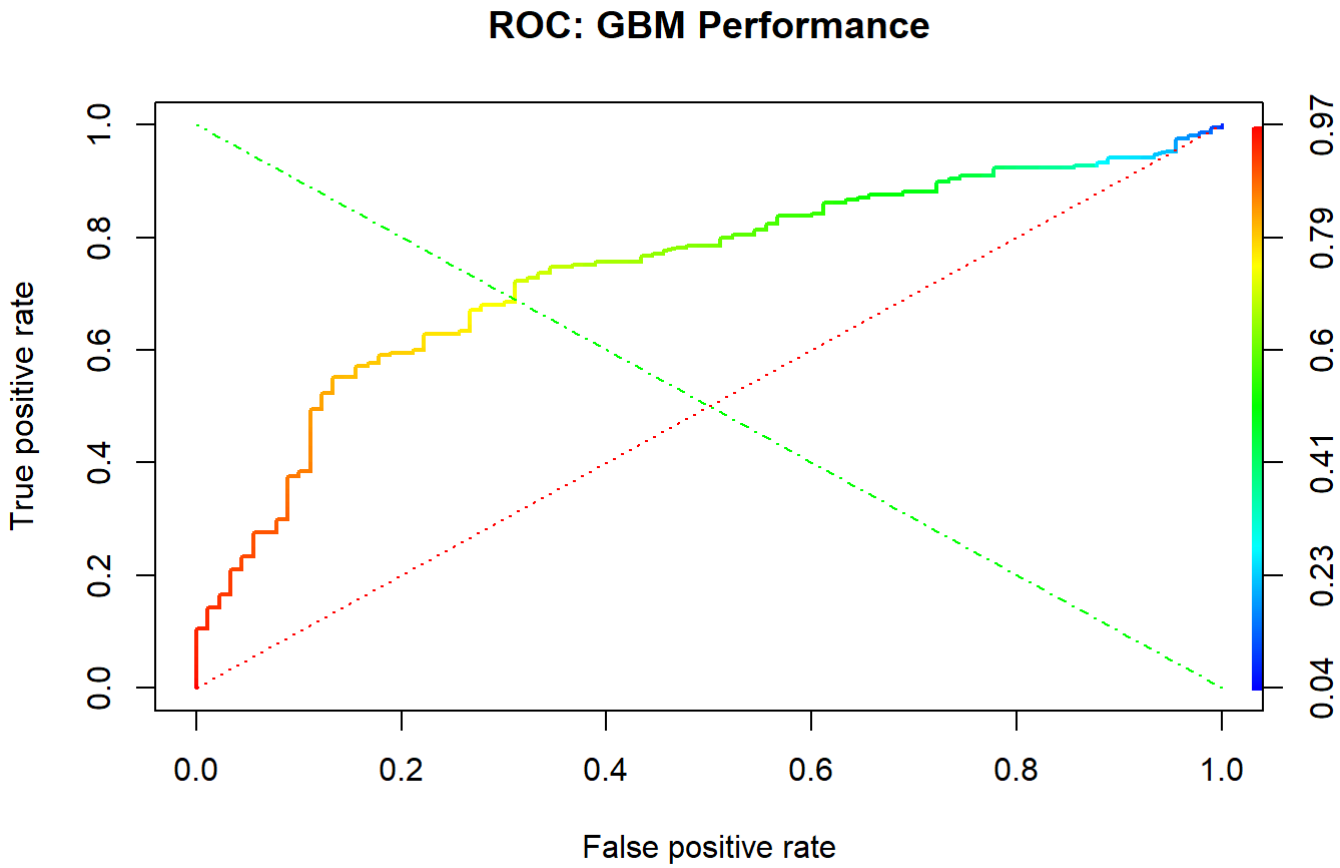



```
# Se emplea type = "response" para convertir los valores predichos en probabilidades.

gbm_score2 <- predict(object = gbm.fit.final2,
                      newdata = test,
                      n.trees = 19,
                      type = "response")

head(gbm_score2)
```

```
## [1] 0.7580310 0.9157374 0.9622986 0.5601250 0.4325598 0.8151039
```

Paso 5. Curva ROC

```
pred_gbm2 <- prediction(gbm_score2, test$target)
perf_gbm2 <- performance(pred_gbm2,"tpr","fpr")
#library(ROCR)
plot(perf_gbm2, 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 6. 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_score2 > 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_score2 > 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_score2 > 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_score2 > 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_score2 > 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] 80.64516 80.74534 81.68421 80.87912 79.24528
```

Se puede observar que el límite donde se mazimiza la F1 es en 0,4, con un F1 = 81.68421

Paso 7. Métricas definitivas

```
#Matriz de confusión con umbral final
score4 <- ifelse(gbm_score2 > 0.40, "Good", "Bad")
MC <- table(test$target, score4)
gbm_Acc_T2 <- round((MC[1,1] + MC[2,2]) / sum(MC) *100, 2)
gbm_Sen_T2 <- round(MC[2,2] / (MC[2,2] + MC[1,2]) *100, 2)
gbm_Pr_T2 <- round(MC[2,2] / (MC[2,2] + MC[2,1]) *100, 2)
gbm_F1_T2 <- round(2*gbm_Pr_T2*gbm_Sen_T2/(gbm_Pr_T2+gbm_Sen_T2), 2)

#KS & AUC
gbm_KS_T2 <- round(max(attr(perf_gbm2,'y.values')[[1]]-attr(perf_gbm2,'x.values')[[1
]])*100, 2)
gbm_AUROC_T2 <- round(performance(pred_gbm2, measure = "auc")@y.values[[1]]*100, 2)

#Métricas finales del modelo
cat("Acierto_gbm: ", gbm_Acc_T2,"\tSensibilidad_gbm: ", gbm_Sen_T2, "\tPrecision_gbm:"
, gbm_Pr_T2, "\tF1-gbm:", gbm_F1_T2, "\tAUROC_gbm: ",gbm_AUROC_T2,"\tKS_gbm: ", gbm_KS
_T2, "\n")
```

```
## Acierto_gbm:  71     Sensibilidad_gbm:  73.21    Precision_gbm: 92.38    F1-gbm: 8
1.69   AUROC_gbm:  73.97   KS_gbm:  41.9
```

Se obtiene una AUC de 73.97, lo que indica un modelo moderadamente aceptable.

6. Comparación de los tres modelos

Comparamos los dos modelos (T1: con 14 variables, y T2 con 12 variables).

```
# Etiquetas de filas
models <- c('GBM_T1', 'GBM_T2')

#Accuracy
models_Acc <- c(gbm_Acc_T1, gbm_Acc_T2)

#Sensibilidad
models_Sen <- c(gbm_Sen_T1, gbm_Sen_T2)

#Precisión
models_Pr <- c(gbm_Pr_T1, gbm_Pr_T2)

#F1
models_F1 <- c(gbm_F1_T1, gbm_F1_T2)

# AUC
models_AUC <- c(gbm_AUROC_T1, gbm_AUROC_T2)

# KS
models_KS <- c(gbm_KS_T1, gbm_KS_T2)
```

```
# 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_T1	74.33	77.14	90	83.08	76.48	44.44
GBM_T2	71	73.21	92.38	81.69	73.97	41.9

Observando la métricade AUC, el primer modelo con 14 variables predictoras resulta mejor (AUC = 76.48) que el segundo modelo con 12 variables (AUC = 73.97).