Funnel L2S Model

Bernardo Lares

22/3/2017

Contents

Modelo Lead To Sale (L2S) . . . 1
Sampling . . . 1
Limpieza y preparación de la data . . . 5
Creación del Modelo . . . 9
Train y Test Data . . . 9
One Hot Encoding para trabajar con XGBoost . . . 9
Parámetros iniciales para el modelo . . . 10
Entrenamiento del modelo . . . 12
Evaluación del modelo . . . 14
Gráficas para visualizar los resultados . . . 17
Exportación del modelo . . . 25

Modelo Lead To Sale (L2S)

En este documento resumiré los pasos seguidos para la generación del score generado con el **Funnel L2S Model** de X empresa, en el cual logramos calcular cuál es la probabilidad de que un lead o una oportunidad nueva vaya a convertir (comprar una póliza todo riesgo) y poder tomar mejores decisiones basadas en Machine Learning.

NOTA:Hay que tomar en cuenta que este score no varía con el tiempo ya que sólo tiene información referente a la persona como tal y no a su comportamiento.

El algorítmo definitivo empleado para generar los scores fue **XGBoost**, utilizando R (sampling, limpieza de data y modelo) y Python (implementación en el CRM).

Sampling

Las librerías empleadas fueron las siguientes:

```
library(RPostgreSQL)
library(dplyr)
library(stringr)
library(Amelia)
library(ggplot2)
library(gridExtra)
```

Luego, se trajo la data del servidor a R directamente usando **PostgreSQL** (se omite el código por motivos de seguridad). Las consultas realizadas de una tabla compuesta por las siguientes dos categorías:

1. **Emitidas (1)**: toda la data de todas las oportunidades que fueron emitidas entre el '2016-01-01' y '2017-01-27' (3 semanas antes de la generación del sample).
2. **No emitidas (0)**: selección aleatoria de oportuniades, dentro del mismo rango de fechas que no fueron emitidas, limitado a 5000.

```
## EMITIDAS
q <- "
SELECT
DISTINCT(o.id),
random(),
o.created,
app.vehicle_body,
app.sex,
EXTRACT(year from app.date_of_birth) as year_of_birth,
app.vehicle_model,
app.vehicle_city,
app.current_situation,
app.vehicle_is_mine,
app.form,
app.already_insured_soat,
app.when_need_policy,
app.vehicle_financed,
app.vehicle_commercial_value,
app.identification,
app.vehicle_is_zero_km,
app.vehicle_has_registration,
app.client_type,
app.vehicle_service_type,
CASE WHEN app.already_insured_with_company IS NOT NULL THEN 'SI'
  ELSE 'NO' END as already_insured,
m.medium,
app.vehicle_brand,
o.quoted_policies_count

FROM applications_carinsuranceapplication as app
LEFT JOIN opportunities_opportunity as o ON (app.id = o.application_object_id)
LEFT JOIN opportunities_userjourney as uj ON (o.id = uj.opportunity_id)
LEFT JOIN opportunities_userjourneystepdone as sd ON (uj.id = sd.user_journey_id)
LEFT JOIN marketing_visitor as m ON (o.marketing_id = m.id)

WHERE
sd.name IN ('issue')
AND o.created BETWEEN '2016-01-01' AND (CURRENT_DATE - INTERVAL '1 month')
AND o.fake = FALSE
AND m.medium != 'api'
AND app.form NOT IN ('default','ux31')
AND o.status != 'descartada'
ORDER BY random()
LIMIT 10000"
q <- dbSendQuery(con, q)
emitidos <- fetch(q, n = -1)
emitidos$emitido <- c(1)
emitidos <- head(emitidos,6000) #Cuántos registros máximos queremos para "1s"

## NO EMITIDAS
q <- "
SELECT
DISTINCT(o.id),
```

```
random(),
o.created,
app.vehicle_body,
app.sex,
EXTRACT(year from app.date_of_birth) as year_of_birth,
app.vehicle_model,
app.vehicle_city,
app.current_situation,
app.vehicle_is_mine,
app.form,
app.already_insured_soat,
app.when_need_policy,
app.vehicle_financed,
app.vehicle_commercial_value,
app.identification,
app.vehicle_is_zero_km,
app.vehicle_has_registration,
app.client_type,
app.vehicle_service_type,
CASE WHEN app.already_insured_with_company IS NOT NULL THEN 'SI'
  ELSE 'NO' END as already_insured,
m.medium,
app.vehicle_brand,
o.quoted_policies_count

FROM applications_carinsuranceapplication as app
LEFT JOIN opportunities_opportunity as o ON (app.id = o.application_object_id)
LEFT JOIN opportunities_userjourney as uj ON (o.id = uj.opportunity_id)
LEFT JOIN opportunities_userjourneystepdone as sd ON (uj.id = sd.user_journey_id)
LEFT JOIN marketing_visitor as m ON (o.marketing_id = m.id)

WHERE
sd.name NOT IN ('issue','terms','payment','docs-physics','acquired','fin')
AND o.created BETWEEN '2016-01-01' AND (CURRENT_DATE - INTERVAL '1 month')
AND o.fake = FALSE
AND m.medium != 'api'
AND app.form NOT IN ('default','ux31')
AND o.status != 'descartada'
ORDER BY random()
LIMIT 8000"
q <- dbSendQuery(con, q)
no.emitidos <- fetch(q, n = -1)
no.emitidos <- filter(no.emitidos,!id %in% emitidos$id)
no.emitidos$emitido <- c(0)
no.emitidos <- head(no.emitidos,5000) #Cuántos registros máximos queremos para "0s"

## JOIN BOTH
sample <- rbind(emitidos,no.emitidos)
## Delete random()
sample$random <- NULL
#Reordenar
sample <- cbind(select(sample,emitido,id,created),select(sample,-emitido,-id,-created))
```

```
## EXPORT RAW
write.csv2(sample,"Data.16.raw.csv")
```

Ahora vemos la data que exportamos del servidor a un archivo CSV:

```
str(sample)
```

```
## 'data.frame':    8545 obs. of  24 variables:
##  $ emitido                : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ id                     : int  518519 415896 394003 441445 556253 516129 525077 474405 551581 330
##  $ created                : POSIXct, format: "2016-12-29 11:35:56" "2016-08-17 08:43:28" ...
##  $ vehicle_body           : chr  "MOTO" "CAMIONETA" "AUTOMOVIL" "AUTOMOVIL" ...
##  $ sex                    : chr  "M" "M" "M" "M" ...
##  $ year_of_birth          : num  1979 1988 1977 1984 1977 ...
##  $ vehicle_model          : int  2015 2017 2015 2004 2016 2008 2015 2015 2014 2016 ...
##  $ vehicle_city           : chr  "Bogotá, Bogota D.C., Colombia" "Cali, Valle del Cauca, Colombia" "
##  $ current_situation      : chr  "" "" "" "" ...
##  $ vehicle_is_mine        : chr  NA NA NA NA ...
##  $ form                   : chr  "shorty" "shorty" "uj40" "shorty" ...
##  $ already_insured_soat   : chr  "only-have-soat" "only-have-soat" "also-i-need-soat" "only-have-soa
##  $ when_need_policy       : chr  "inmediately" "inmediately" "in_a_week_or_less" "inmediately" ...
##  $ vehicle_financed       : chr  NA NA NA NA ...
##  $ vehicle_commercial_value: num  2.68e+07 1.04e+08 2.70e+07 1.16e+07 8.31e+07 ...
##  $ identification         : chr  "80009087" "1024463624" "94472796" "8126920" ...
##  $ vehicle_is_zero_km     : int  0 1 0 0 0 0 0 0 0 0 ...
##  $ vehicle_has_registration: int  1 0 0 1 1 1 1 1 1 1 ...
##  $ client_type            : chr  "natural" "natural" "natural" "natural" ...
##  $ vehicle_service_type   : chr  "particular" "publico" "particular" "particular" ...
##  $ already_insured        : chr  "NO" "NO" "NO" "NO" ...
##  $ medium                 : chr  "direct" "direct" "cpc" "cpc" ...
##  $ vehicle_brand          : chr  "YAMAHA" "RENAULT" "CHEVROLET" "CHEVROLET" ...
##  $ quoted_policies_count  : int  1 1 14 10 14 10 10 1 6 17 ...
```

```
head(sample,1)
```

```
##   emitido     id             created vehicle_body sex year_of_birth
## 1       1 518519 2016-12-29 11:35:56         MOTO   M          1979
##   vehicle_model                  vehicle_city current_situation
## 1          2015 Bogotá, Bogota D.C., Colombia
##   vehicle_is_mine   form already_insured_soat when_need_policy
## 1            <NA> shorty       only-have-soat      inmediately
##   vehicle_financed vehicle_commercial_value identification
## 1             <NA>                 26800000       80009087
##   vehicle_is_zero_km vehicle_has_registration client_type
## 1                  0                        1     natural
##   vehicle_service_type already_insured medium vehicle_brand
## 1           particular              NO direct        YAMAHA
##   quoted_policies_count
## 1                     1
```

```
sample %>% group_by(emitido) %>% tally()
```

```
## # A tibble: 2 × 2
##   emitido     n
##     <dbl> <int>
## 1       0  5000
```

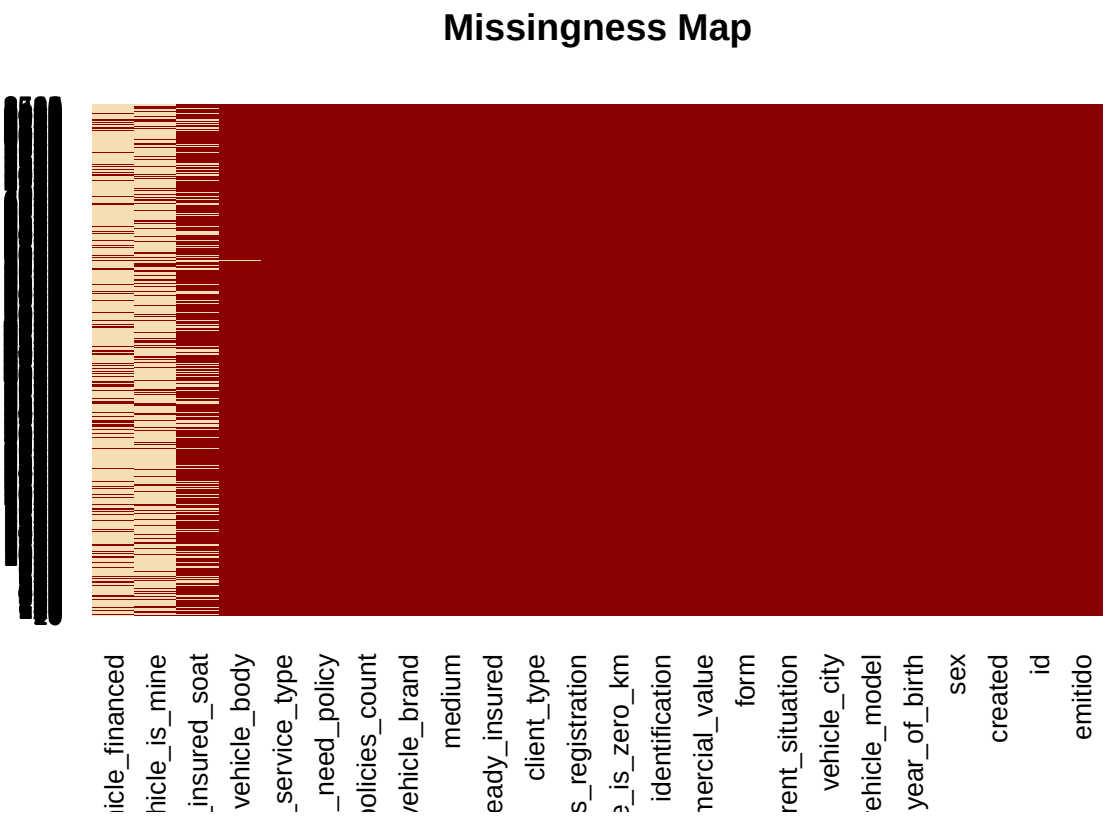
```
## 2       1  3545
```

Limpieza y preparación de la data

Ahora que tenemos ya el extracto de la data que usaremos para el modelo, preparamos los campos que requieran organizar, limpiar, mejorar...

```
df.train <- read.csv2('Data.16.raw.csv',stringsAsFactors=TRUE)
df.train$X <- NULL #No nos sirve
df.train$civil_status <- NULL #Mala data
df.train$email_address <- NULL #Mal predictor
df.train$domain <- NULL #Mal predictor
df.train$mobile_phone <- NULL #Mal predictor
df.train$phone <- NULL #Mal predictor

#Missing values
missmap(df.train,legend=FALSE,rank.order=TRUE)
```




```
#CIUDADES
#Fix data
df.train$vehicle_city <- sub(", Colombia","",df.train$vehicle_city)
df.train$vehicle_city <- sub(", .*","",df.train$vehicle_city)
#View data - Ciudad
fix <- as.data.frame(df.train$vehicle_city)
fix <- fix %>% group_by(Campo=fix[,1]) %>% tally(sort = TRUE) %>%
  mutate(Perc=round(n/nrow(fix),2)) %>% top_n(10,n)
print(fix)
```

```
## # A tibble: 10 × 3
##           Campo     n  Perc
##          <fctr> <int> <dbl>
## 1        Bogotá  4217  0.49
## 2      Medellín  1684  0.20
## 3          Cali   929  0.11
## 4  Barranquilla   302  0.04
## 5   Bucaramanga   146  0.02
## 6       Pereira   121  0.01
## 7     Cartagena    99  0.01
## 8        Ibagué    89  0.01
## 9     Manizales    87  0.01
## 10      Armenia    64  0.01
```

```
#Reescribir las campos por nueva agrupación
df.train <- df.train %>%
  mutate(vehicle_city = ifelse(vehicle_city=="Bogotá","BOG",
                               ifelse(vehicle_city=="Medellín","MED",
                                      ifelse(vehicle_city=="Cali","CAL",
                                             ifelse(vehicle_city=="Barranquilla","BAR","OTRA")))))

# NACIMIENTO - EDAD
df.train$year_of_birth[is.na(df.train$year_of_birth)] <- as.integer(format(Sys.time(), "%Y")) #NAs
df.train$edad <- as.integer(format(Sys.time(), "%Y")) - df.train$year_of_birth #Edades
df.train$year_of_birth <- NULL
df.train <- df.train %>%
  mutate(edad = ifelse(edad>=80,">80",
                       ifelse(edad>=55,"55-79",
                              ifelse(edad>=40,"40-54",
                                     ifelse(edad>=30,"30-39",
                                            ifelse(edad>=18,"18-29",
                                                   ifelse(edad>=1,"MENOR","SIN")))))))

# MODELO DEL VEHÍCULO
fix <- as.data.frame(df.train$vehicle_model)
fix <- fix %>% group_by(Campo=fix[,1]) %>% tally(sort = TRUE) %>%
  mutate(Perc=round(n/nrow(fix),2)) %>% top_n(10,n)
print(fix)
```

```
## # A tibble: 10 × 3
##    Campo     n  Perc
##    <int> <int> <dbl>
## 1   2016  1032  0.12
## 2   2015  1018  0.12
## 3   2013   877  0.10
## 4   2017   823  0.10
## 5   2012   801  0.09
## 6   2011   763  0.09
## 7   2014   749  0.09
## 8   2008   462  0.05
## 9   2010   431  0.05
## 10  2009   393  0.05
```

```
#Reescribir las campos por nueva agrupación
año <- as.integer(format(Sys.Date(), "%Y"))
```

```
df.train <- df.train %>%
  mutate(vehicle_model = ifelse(vehicle_model>=año,"DEL.AÑO",
                                ifelse(vehicle_model>=(año-1),"AÑO.PASADO",
                                       ifelse(vehicle_model>=(año-2),"AÑO.ANTEPASADO",
                                              ifelse(vehicle_model>=(año-5),"5.AÑOS",
                                                     ifelse(vehicle_model>=(año-10),"10.AÑOS","MAS.10.A

# MEDIUM
fix <- as.data.frame(df.train$medium)
fix <- fix %>% group_by(Campo=fix[,1]) %>% tally(sort = TRUE) %>%
  mutate(Perc=round(n/nrow(fix),2)) %>% top_n(10,n)
print(fix)
```

```
## # A tibble: 10 × 3
##                   Campo     n  Perc
##                  <fctr> <int> <dbl>
## 1                   cpc  3798  0.44
## 2                direct  1807  0.21
## 3                   seo  1710  0.20
## 4                    ET   829  0.10
## 5              referral   231  0.03
## 6     link-calculadora    59  0.01
## 7               autolab    18  0.00
## 8           EXACTTARGET    18  0.00
## 9  facilidades de pago     9  0.00
## 10              DISPLAY     8  0.00
```

```
#Reescribir las campos por nueva agrupación
df.train <- df.train %>%
  mutate(medium = ifelse(medium=="cpc","SEM",
                         ifelse(medium=="direct","DIRECT",
                                ifelse(medium=="seo","SEO",
                                       ifelse(medium=="et","ET",
                                              ifelse(medium=="referral","REFERRAL",
                                                     ifelse(medium=="INBOXLABS","INBOXLABS","OTRO")))))

# BODY
fix <- as.data.frame(df.train$vehicle_body)
fix <- fix %>% group_by(Campo=fix[,1]) %>% tally(sort = TRUE) %>%
  mutate(Perc=round(n/nrow(fix),2)) %>% top_n(10,n)
print(fix)
```

```
## # A tibble: 10 × 3
##                Campo     n  Perc
##               <fctr> <int> <dbl>
## 1          AUTOMOVIL  5907  0.69
## 2          CAMIONETA  1477  0.17
## 3               MOTO   295  0.03
## 4            CAMPERO   203  0.02
## 5  CAMIONETA PASAJ.   199  0.02
## 6             PESADO   135  0.02
## 7             PICKUP   112  0.01
## 8  PICKUP DOBLE CAB    78  0.01
## 9        MOTOCICLETA    43  0.01
```

```
## 10              BUS    34  0.00
```

```
#Reescribir las campos por nueva agrupación
df.train <- df.train %>%
  mutate(vehicle_body = ifelse(vehicle_body=="AUTOMOVIL","AUTOMOVIL",
                               ifelse(vehicle_body=="CAMIONETA","CAMIONETA",
                                      ifelse(vehicle_body=="MOTO","MOTO",
                                             ifelse(vehicle_body=="CAMPERO","CAMPERO",
                                                    ifelse(vehicle_body=="CAMIONETA PASAJ.","CAMIONETA
                                                           ifelse(vehicle_body=="PICKUP","PICKUP","OTRO

# IDENTIFICACIÓN
fix <- as.data.frame(df.train$identification)
fix <- fix %>% group_by(Campo=fix[,1]) %>% tally(sort = TRUE) %>%
  mutate(Perc=round(n/nrow(fix),2)) %>% top_n(5,n)
print(fix)
```

```
## # A tibble: 5 × 3
##        Campo     n  Perc
##       <fctr> <int> <dbl>
## 1 0000000001  1407  0.16
## 2 1130658530     5  0.00
## 3  000000001     4  0.00
## 4 1018406341     4  0.00
## 5 1024558143     4  0.00
```

```
#Reescribir las campos por nueva agrupación
df.train <- df.train %>% mutate(identification = ifelse(as.integer(identification)==1,0,1))
df.train$identification[is.na(df.train$identification)] <- 0

# CREATED (Día de semana)
df.train$weekday <- weekdays(as.Date(df.train$created,format='%Y-%m-%d',tz="BO"))

#Clases de los campos
for(i in c(1,1:ncol(df.train))) {
  df.train[,i] <- as.factor(df.train[,i])
}
df.train[,2] <- as.integer(as.character(df.train[,2]))

## EXPORT CLEAN
write.csv2(df.train,"Data.16.clean.csv")
str(df.train)
```

```
## 'data.frame':    8545 obs. of  25 variables:
##  $ emitido                : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
##  $ id                     : int  518519 415896 394003 441445 556253 516129 525077 474405 551581 330
##  $ created                : Factor w/ 8375 levels "2016-01-04 12:11:00",..: 6134 1706 988 2788 8263
##  $ vehicle_body           : Factor w/ 7 levels "AUTOMOVIL","CAMIONETA",..: 5 2 1 1 2 4 1 2 6 1 ...
##  $ sex                    : Factor w/ 3 levels "","F","M": 3 3 3 3 3 3 3 2 3 3 ...
##  $ vehicle_model          : Factor w/ 6 levels "10.AÑOS","5.AÑOS",..: 3 5 3 6 4 1 3 3 2 4 ...
##  $ vehicle_city           : Factor w/ 5 levels "BAR","BOG","CAL",..: 2 3 3 4 2 4 2 5 1 1 ...
##  $ current_situation      : Factor w/ 7 levels "","asociados",..: 1 1 1 1 1 1 1 1 1 6 ...
##  $ vehicle_is_mine        : Factor w/ 2 levels "no","yes": NA NA NA NA NA 2 NA NA 2 1 ...
##  $ form                   : Factor w/ 5 levels "rastreator-v2",..: 2 2 3 2 2 3 2 3 2 5 ...
##  $ already_insured_soat   : Factor w/ 4 levels "also-i-need-soat",..: 4 4 1 4 4 1 4 4 4 NA ...
##  $ when_need_policy       : Factor w/ 5 levels "asociados","between_one_and_two_weeks",..: 5 5 4 5 4
```

```
##  $ vehicle_financed        : Factor w/ 3 levels "no-use-savings",..: NA NA NA NA NA NA NA NA NA NA .
##  $ vehicle_commercial_value: Factor w/ 1160 levels "220000","950000",..: 275 1006 277 114 897 342 288
##  $ identification          : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
##  $ vehicle_is_zero_km      : Factor w/ 2 levels "0","1": 1 2 1 1 1 1 1 1 1 1 ...
##  $ vehicle_has_registration: Factor w/ 2 levels "0","1": 2 1 1 2 2 2 2 2 2 2 ...
##  $ client_type             : Factor w/ 2 levels "juridica","natural": 2 2 2 2 2 2 2 2 1 2 ...
##  $ vehicle_service_type    : Factor w/ 4 levels "","particular",..: 2 3 2 2 2 2 2 3 2 2 ...
##  $ already_insured         : Factor w/ 2 levels "NO","SI": 1 1 1 1 1 2 1 1 1 1 ...
##  $ medium                  : Factor w/ 6 levels "DIRECT","INBOXLABS",..: 1 1 5 5 6 6 5 5 1 3 ...
##  $ vehicle_brand           : Factor w/ 85 levels "AKT","ALFA ROMEO",..: 82 64 14 14 34 18 14 13 28 14
##  $ quoted_policies_count   : Factor w/ 30 levels "0","1","2","3",..: 2 2 15 11 15 11 11 2 7 18 ...
##  $ edad                    : Factor w/ 5 levels "18-29","30-39",..: 2 1 3 2 3 2 3 1 5 1 ...
##  $ weekday                 : Factor w/ 7 levels "Friday","Monday",..: 5 7 4 7 2 2 1 1 2 1 ...
```

Creación del Modelo

Una vez tenemos la data lista para nuestro modelo, podemos entrenarlo, ajustarlo y conseguir los mejores resultados.

Las librerías empleadas son las siguientes:

```
library(dplyr)
library(xgboost)
library(data.table)
library(caTools)
library(pROC)
library(gridExtra)
library(caret)
library(ggplot2)
```

Importamos la data limpia para el modelo:

```
df.train <- read.csv2('Data.16.clean.csv',stringsAsFactors=TRUE,na=as.factor("NULO"))
fechas <- paste(min(as.Date(df.train$created)),"-",max(as.Date(df.train$created)))
df.train$X <- NULL
df.train$vehicle_commercial_value <- as.numeric(as.character(df.train$vehicle_commercial_value))
df.train$vehicle_brand <- NULL #Noice
```

Train y Test Data

Dividimos la data en train (para entrenamiento) y test (para las pruebas), con una relación de 30/70.

```
set.seed(1)
split <- sample.split(df.train$emitido, SplitRatio = 0.7)
train <- subset(df.train, split == TRUE) #Training
test <- subset(df.train, split == FALSE) #Testing

n <- nrow(test)
emitidos <- nrow(filter(test,emitido==1))
```

One Hot Encoding para trabajar con XGBoost

Una vez tengamos nuestra data segmentada, la preparamos para XGBoost:

```
setDT(train)
setDT(test)

#One hot encoding
labels <- train$emitido #Target train
ts_label <- test$emitido #Target test
new_tr <- model.matrix(~.+0,data = train[,-c("emitido","id","created"),with=F])
new_ts <- model.matrix(~.+0,data = test[,-c("emitido","id","created"),with=F])

#Convert data table into a matrix (xgb.DMatrix):
dtrain <- xgb.DMatrix(data = new_tr,label = labels)
dtest <- xgb.DMatrix(data = new_ts,label = ts_label)
```

Parámetros iniciales para el modelo

Definimos los valores de los parámetros para iniciar el modelo:

```
params <- list(
  booster = "gbtree", # 'gbtree' / 'gblinear'
  objective = "binary:logistic", # 'binary:logistic' / 'reg:linear'
  eta=0.1, #Step size shrinkage (prevents overfitting) - default=0.3
  gamma=0, #Minimum loss reduction required to split
  max_depth=5, #Default=6 <- Complexity (ver xgb.plot.deepness)
  min_child_weight=1,
  subsample=1,#Robust to noise
  colsample_bytree=1 #Robust to noise
)
```

Hacemos cross-validation buscando la mejor iteración para este modelo. Además, podemos calcular el Accuracy del cross-validation.

```
xgbcv <- xgb.cv(params = params,
                data = dtrain,
                nrounds = 150, #n iteraciones
                nfold = 5, #folds cross validation
                showsd = T,
                stratified = T,
                print_every_n = 1, #Intervalos a mostrar
                early_stopping_rounds = 20, #20
                maximize = F,
                prediction = F)
```

```
## [1]  train-error:0.321171+0.004813   test-error:0.328984+0.012156
## Multiple eval metrics are present. Will use test_error for early stopping.
## Will train until test_error hasn't improved in 20 rounds.
##
## [2]  train-error:0.319917+0.005066   test-error:0.328817+0.011781
## [3]  train-error:0.315780+0.004776   test-error:0.325305+0.010111
## [4]  train-error:0.319249+0.007187   test-error:0.329652+0.010275
## [5]  train-error:0.313816+0.005039   test-error:0.326645+0.007756
## [6]  train-error:0.315363+0.004695   test-error:0.332327+0.011804
## [7]  train-error:0.311685+0.004681   test-error:0.328819+0.011080
## [8]  train-error:0.307882+0.003074   test-error:0.321294+0.008103
## [9]  train-error:0.306962+0.003189   test-error:0.319287+0.010362
```

```
## [10] train-error:0.304706+0.002174   test-error:0.318452+0.008816
## [11] train-error:0.302282+0.002198   test-error:0.314105+0.008427
## [12] train-error:0.302366+0.001974   test-error:0.314105+0.007766
## [13] train-error:0.301362+0.003040   test-error:0.312768+0.010443
## [14] train-error:0.299147+0.003155   test-error:0.311097+0.010231
## [15] train-error:0.299064+0.002864   test-error:0.310763+0.009411
## [16] train-error:0.296055+0.003618   test-error:0.310428+0.011574
## [17] train-error:0.294383+0.003302   test-error:0.307921+0.011026
## [18] train-error:0.293422+0.003182   test-error:0.306917+0.011433
## [19] train-error:0.291876+0.002240   test-error:0.307920+0.010444
## [20] train-error:0.291625+0.002294   test-error:0.307084+0.010418
## [21] train-error:0.290120+0.003395   test-error:0.304409+0.008342
## [22] train-error:0.288658+0.002046   test-error:0.304409+0.006939
## [23] train-error:0.287696+0.002236   test-error:0.304911+0.006886
## [24] train-error:0.286861+0.002116   test-error:0.301399+0.008154
## [25] train-error:0.285816+0.002246   test-error:0.301901+0.007571
## [26] train-error:0.284395+0.000888   test-error:0.300563+0.010059
## [27] train-error:0.283810+0.001155   test-error:0.300730+0.009804
## [28] train-error:0.281637+0.001209   test-error:0.300898+0.009690
## [29] train-error:0.281428+0.001154   test-error:0.300229+0.009465
## [30] train-error:0.280257+0.001446   test-error:0.301065+0.009736
## [31] train-error:0.278711+0.002076   test-error:0.299393+0.008839
## [32] train-error:0.278168+0.001596   test-error:0.299393+0.009307
## [33] train-error:0.276956+0.001319   test-error:0.298391+0.008247
## [34] train-error:0.276329+0.001714   test-error:0.299059+0.009584
## [35] train-error:0.276036+0.001283   test-error:0.299061+0.008700
## [36] train-error:0.275911+0.001415   test-error:0.299228+0.008015
## [37] train-error:0.274197+0.002870   test-error:0.299060+0.007879
## [38] train-error:0.272735+0.002776   test-error:0.296218+0.008234
## [39] train-error:0.271230+0.002733   test-error:0.298223+0.009333
## [40] train-error:0.270938+0.002844   test-error:0.297721+0.009824
## [41] train-error:0.270604+0.002712   test-error:0.298223+0.009686
## [42] train-error:0.269643+0.002539   test-error:0.298055+0.009750
## [43] train-error:0.269475+0.002596   test-error:0.297387+0.009763
## [44] train-error:0.268765+0.002763   test-error:0.297888+0.010169
## [45] train-error:0.268096+0.002493   test-error:0.298891+0.010473
## [46] train-error:0.267302+0.002875   test-error:0.298054+0.010888
## [47] train-error:0.265714+0.003432   test-error:0.298222+0.010614
## [48] train-error:0.265296+0.003356   test-error:0.298055+0.009812
## [49] train-error:0.265547+0.003315   test-error:0.297219+0.009869
## [50] train-error:0.263123+0.004959   test-error:0.297554+0.008427
## [51] train-error:0.263332+0.004861   test-error:0.296217+0.008935
## [52] train-error:0.262872+0.004324   test-error:0.296049+0.009948
## [53] train-error:0.262412+0.004297   test-error:0.297387+0.008563
## [54] train-error:0.261995+0.004815   test-error:0.296217+0.007958
## [55] train-error:0.261493+0.004910   test-error:0.296217+0.007599
## [56] train-error:0.260448+0.003586   test-error:0.296050+0.008403
## [57] train-error:0.259404+0.003556   test-error:0.295716+0.008282
## [58] train-error:0.259404+0.003953   test-error:0.295548+0.008844
## [59] train-error:0.259111+0.004004   test-error:0.295381+0.007751
## [60] train-error:0.259069+0.003571   test-error:0.295382+0.008087
## [61] train-error:0.256938+0.003015   test-error:0.297722+0.007785
## [62] train-error:0.256520+0.003866   test-error:0.296886+0.008336
## [63] train-error:0.255935+0.003757   test-error:0.297220+0.008310
```

```
## [64] train-error:0.255726+0.003709   test-error:0.298055+0.009864
## [65] train-error:0.255433+0.003303   test-error:0.296550+0.010148
## [66] train-error:0.254514+0.003937   test-error:0.296885+0.009715
## [67] train-error:0.253720+0.003646   test-error:0.296550+0.010056
## [68] train-error:0.252257+0.003492   test-error:0.297218+0.011275
## [69] train-error:0.251170+0.003893   test-error:0.298054+0.010981
## [70] train-error:0.250418+0.003672   test-error:0.296382+0.011985
## [71] train-error:0.251087+0.003604   test-error:0.296216+0.011299
## [72] train-error:0.249875+0.004019   test-error:0.295381+0.010129
## [73] train-error:0.249206+0.003986   test-error:0.295381+0.009590
## [74] train-error:0.248914+0.003849   test-error:0.296216+0.010101
## [75] train-error:0.248705+0.004037   test-error:0.296049+0.009957
## [76] train-error:0.248161+0.003452   test-error:0.295046+0.010907
## [77] train-error:0.247451+0.003508   test-error:0.295214+0.010023
## [78] train-error:0.246239+0.003733   test-error:0.295046+0.009832
## [79] train-error:0.245695+0.003422   test-error:0.295548+0.010340
## [80] train-error:0.245194+0.003626   test-error:0.297053+0.010609
## [81] train-error:0.245737+0.003347   test-error:0.296384+0.009996
## [82] train-error:0.245152+0.002756   test-error:0.295214+0.010110
## [83] train-error:0.244567+0.003375   test-error:0.295882+0.010659
## [84] train-error:0.243439+0.003228   test-error:0.296050+0.009318
## [85] train-error:0.242561+0.003297   test-error:0.296718+0.009949
## [86] train-error:0.242018+0.003395   test-error:0.296217+0.010525
## [87] train-error:0.241642+0.003409   test-error:0.296384+0.010353
## [88] train-error:0.241433+0.003048   test-error:0.296719+0.011167
## [89] train-error:0.239970+0.003196   test-error:0.297387+0.010174
## [90] train-error:0.239385+0.002807   test-error:0.295883+0.010084
## [91] train-error:0.238842+0.003363   test-error:0.296218+0.009801
## [92] train-error:0.239051+0.003882   test-error:0.295549+0.010033
## [93] train-error:0.238758+0.003702   test-error:0.296050+0.009858
## [94] train-error:0.238048+0.003811   test-error:0.295716+0.010009
## [95] train-error:0.237630+0.004126   test-error:0.296217+0.009427
## [96] train-error:0.237463+0.003865   test-error:0.296050+0.009872
## Stopping. Best iteration:
## [76] train-error:0.248161+0.003452   test-error:0.295046+0.010907
```

```
#The model returned lowest error @:
bestn <- xgbcv$best_iteration #cambia cada vez = 68 empleado en CRM
paste("Mejor iteración:",bestn)
```

```
## [1] "Mejor iteración: 76"
```

```
paste("CV Accuracy: ",round((1-min(xgbcv$evaluation_log$test_error_mean))*100,2),"%",sep="")
```

```
## [1] "CV Accuracy: 70.5%"
```

Entrenamiento del modelo

Y ahora, entrenamos nuestro modelo de pruebas y calculamos Accuracy:

```
xgb1 <- xgb.train(
  params = params,
  data = dtrain,
  nrounds = bestn, #default=bestn para no hacer overfitting
  watchlist = list(val=dtest,train=dtrain),
```

```
  print_every_n = 1,
  maximize = F,
  eval_metric = "error"
)
```

```
## [1]  val-error:0.331643  train-error:0.321464
## [2]  val-error:0.330862  train-error:0.321632
## [3]  val-error:0.326961  train-error:0.315446
## [4]  val-error:0.323449  train-error:0.315446
## [5]  val-error:0.327351  train-error:0.318623
## [6]  val-error:0.321108  train-error:0.315112
## [7]  val-error:0.321888  train-error:0.314611
## [8]  val-error:0.322669  train-error:0.314611
## [9]  val-error:0.317987  train-error:0.308592
## [10] val-error:0.317987  train-error:0.308091
## [11] val-error:0.317206  train-error:0.305751
## [12] val-error:0.314865  train-error:0.304915
## [13] val-error:0.314475  train-error:0.304748
## [14] val-error:0.313695  train-error:0.304246
## [15] val-error:0.314085  train-error:0.301237
## [16] val-error:0.312524  train-error:0.299231
## [17] val-error:0.312524  train-error:0.297392
## [18] val-error:0.313695  train-error:0.296556
## [19] val-error:0.312915  train-error:0.295888
## [20] val-error:0.312524  train-error:0.294885
## [21] val-error:0.312915  train-error:0.293213
## [22] val-error:0.312134  train-error:0.292377
## [23] val-error:0.305892  train-error:0.289034
## [24] val-error:0.305501  train-error:0.288699
## [25] val-error:0.307452  train-error:0.288198
## [26] val-error:0.309013  train-error:0.288031
## [27] val-error:0.303941  train-error:0.284019
## [28] val-error:0.303160  train-error:0.284186
## [29] val-error:0.302380  train-error:0.283852
## [30] val-error:0.298088  train-error:0.283016
## [31] val-error:0.297698  train-error:0.282180
## [32] val-error:0.298869  train-error:0.280843
## [33] val-error:0.298088  train-error:0.280508
## [34] val-error:0.297308  train-error:0.279338
## [35] val-error:0.295357  train-error:0.279505
## [36] val-error:0.295357  train-error:0.279171
## [37] val-error:0.295357  train-error:0.278168
## [38] val-error:0.291455  train-error:0.276329
## [39] val-error:0.293796  train-error:0.275326
## [40] val-error:0.293796  train-error:0.275159
## [41] val-error:0.293406  train-error:0.274992
## [42] val-error:0.295357  train-error:0.274824
## [43] val-error:0.295357  train-error:0.274156
## [44] val-error:0.295357  train-error:0.273654
## [45] val-error:0.297308  train-error:0.274323
## [46] val-error:0.295357  train-error:0.272150
## [47] val-error:0.295747  train-error:0.272484
## [48] val-error:0.294577  train-error:0.272484
## [49] val-error:0.295357  train-error:0.271481
```

```
## [50] val-error:0.294967  train-error:0.272150
## [51] val-error:0.297308  train-error:0.272150
## [52] val-error:0.297308  train-error:0.269141
## [53] val-error:0.298088  train-error:0.268472
## [54] val-error:0.296528  train-error:0.267971
## [55] val-error:0.297698  train-error:0.268472
## [56] val-error:0.297698  train-error:0.266800
## [57] val-error:0.296528  train-error:0.267469
## [58] val-error:0.296918  train-error:0.265965
## [59] val-error:0.296918  train-error:0.266132
## [60] val-error:0.298088  train-error:0.265965
## [61] val-error:0.297698  train-error:0.266633
## [62] val-error:0.296918  train-error:0.266466
## [63] val-error:0.296918  train-error:0.266466
## [64] val-error:0.298869  train-error:0.266132
## [65] val-error:0.298088  train-error:0.265296
## [66] val-error:0.297698  train-error:0.264627
## [67] val-error:0.297698  train-error:0.263791
## [68] val-error:0.296918  train-error:0.263290
## [69] val-error:0.297308  train-error:0.263123
## [70] val-error:0.297698  train-error:0.262120
## [71] val-error:0.298088  train-error:0.261618
## [72] val-error:0.298088  train-error:0.261284
## [73] val-error:0.300819  train-error:0.259278
## [74] val-error:0.300819  train-error:0.259278
## [75] val-error:0.300429  train-error:0.259779
## [76] val-error:0.299649  train-error:0.258609
```

```
paste("Accuracy: ",round((1-min(xgb1$evaluation_log$val_error))*100,2),"%",sep="")
```

```
## [1] "Accuracy: 70.85%"
```

Evaluación del modelo

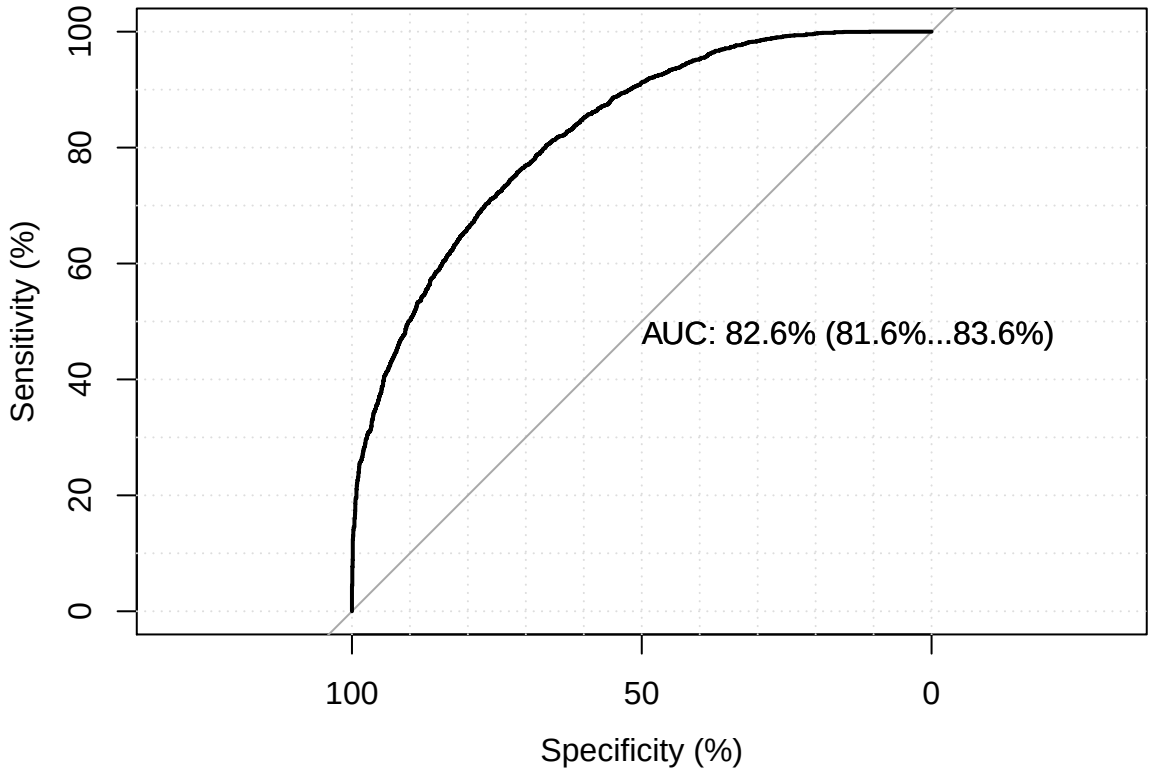
Una vez satisfechos con el valor obtenido de Accuracy, podemos empezar a evaluar, predecir, estudiar los resultados y exportarlo.

```
result <- as.data.frame(cbind(id_opp=train$id,date=as.Date(train$created),real=train$emitido,score=predi
result$date <- as.Date(result$date,origin='1970-01-01')
head(result)
```

```
##   id_opp       date real     score
## 1 518519 2016-12-29    1 0.9470765
## 2 415896 2016-08-17    1 0.8834900
## 3 394003 2016-07-24    1 0.8087590
## 4 556253 2017-02-20    1 0.8186314
## 5 474405 2016-11-04    1 0.8972018
## 6 551581 2017-02-13    1 0.6598039
```

```
threshold <- 0.3 #Definir threshold (0.5 es lo convencional pero depende del caso)
result <- result %>% mutate(predicción=ifelse(score>=threshold,1,0))
xgbpred <- predict(xgb1,dtest)
xgbpred <- ifelse(xgbpred>threshold,1,0)
mat <- xgb.importance(feature_names=colnames(new_tr),model=xgb1) # Importancia variables
MC <- table(test$emitido, xgbpred > threshold)
```

```
deciles <- quantile(result$score, probs = seq(0.1, 0.9, length = 9), names = TRUE)
deciles <- data.frame(cbind(
  Deciles=row.names(as.data.frame(deciles)),
  Threshold=as.data.frame(deciles)),row.names=NULL)
#Entonces, tenemos:
resultados <- list("Mejor iteración + ACC"=
                     paste(
                       max(xgb1$evaluation_log$iter),'<-',
                       round((1-min(xgb1$evaluation_log$val_error))*100,2),"%"),
                   "Top 10 predictores"=mat[1:10,1:2],
                   "Rango de fechas"=fechas,
                   "% Relación Emitidas"=paste(
                     round(emitidos/n,2),"<-",emitidos,"emitidos"),
                   "Matriz de Confusión @Threshold"=MC,
                   "Threshold empleada"=threshold,
                   "Accuracy (ACC) @Threshold"=round((MC[1,1]+MC[2,2])/n,4),
                   "% True Positives: emitida & gestionada"=MC[2,2]/emitidos,
                   "% True: total gestionadas"=(MC[1,2]+MC[2,2])/n,
                   "Curva ROC"=plot.roc(
                     x=result$real,
                     predictor=result$score,
                     smooth=FALSE,auc=TRUE,ci=TRUE,print.auc=TRUE,percent=TRUE,grid=TRUE),
                   "Deciles"=deciles)
```




```
print(resultados)
```

```
## $`Mejor iteración + ACC`
## [1] "76 <- 70.85 %"
##
```

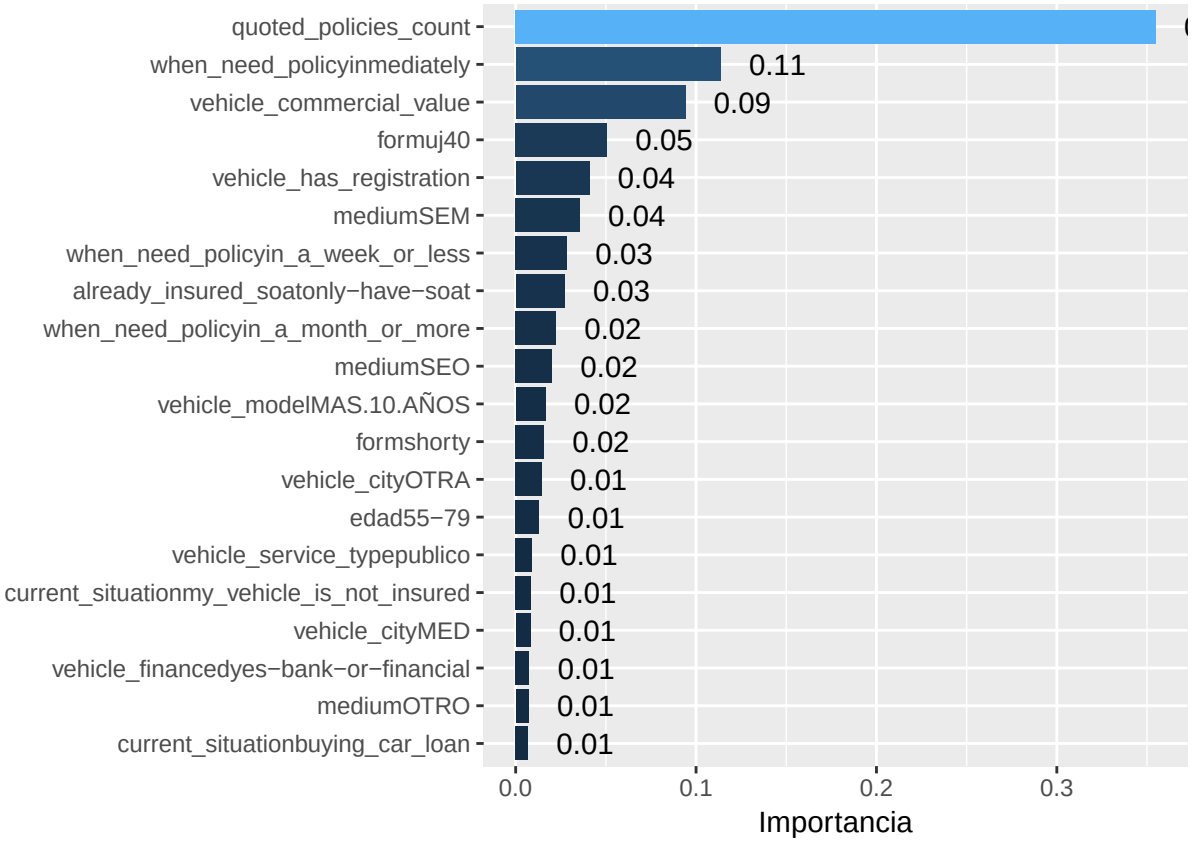
```
## $`Top 10 predictores`
##                                 Feature       Gain
##  1:              quoted_policies_count 0.35483166
##  2:         when_need_policyinmediately 0.11359892
##  3:            vehicle_commercial_value 0.09406018
##  4:                            formuj40 0.05064913
##  5:            vehicle_has_registration 0.04086160
##  6:                           mediumSEM 0.03551133
##  7:  when_need_policyin_a_week_or_less 0.02848036
##  8: already_insured_soatonly-have-soat 0.02714296
##  9: when_need_policyin_a_month_or_more 0.02233710
## 10:                           mediumSEO 0.02017854
##
## $`Rango de fechas`
## [1] "2016-01-04 - 2017-02-22"
##
## $`% Relación Emitidas`
## [1] "0.41 <- 1063 emitidos"
##
## $`Matriz de Confusión @Threshold`
##
##     FALSE TRUE
##   0   673  827
##   1   136  927
##
## $`Threshold empleada`
## [1] 0.3
##
## $`Accuracy (ACC) @Threshold`
## [1] 0.6243
##
## $`% True Positives: emitida & gestionada`
## [1] 0.8720602
##
## $`% True: total gestionadas`
## [1] 0.6843543
##
## $`Curva ROC`
##
## Call:
## plot.roc.default(x = result$real, predictor = result$score, smooth = FALSE,     auc = TRUE, ci = TRU
##
## Data: result$score in 3500 controls (result$real 0) < 2482 cases (result$real 1).
## Area under the curve: 82.58%
## 95% CI: 81.56%-83.6% (DeLong)
##
## $Deciles
##   Deciles   deciles
## 1     10% 0.1199065
## 2     20% 0.2194323
## 3     30% 0.2859238
## 4     40% 0.3491037
## 5     50% 0.4030725
## 6     60% 0.4525616
```

```
## 7      70% 0.5085006
## 8      80% 0.6123303
## 9      90% 0.7216189
```

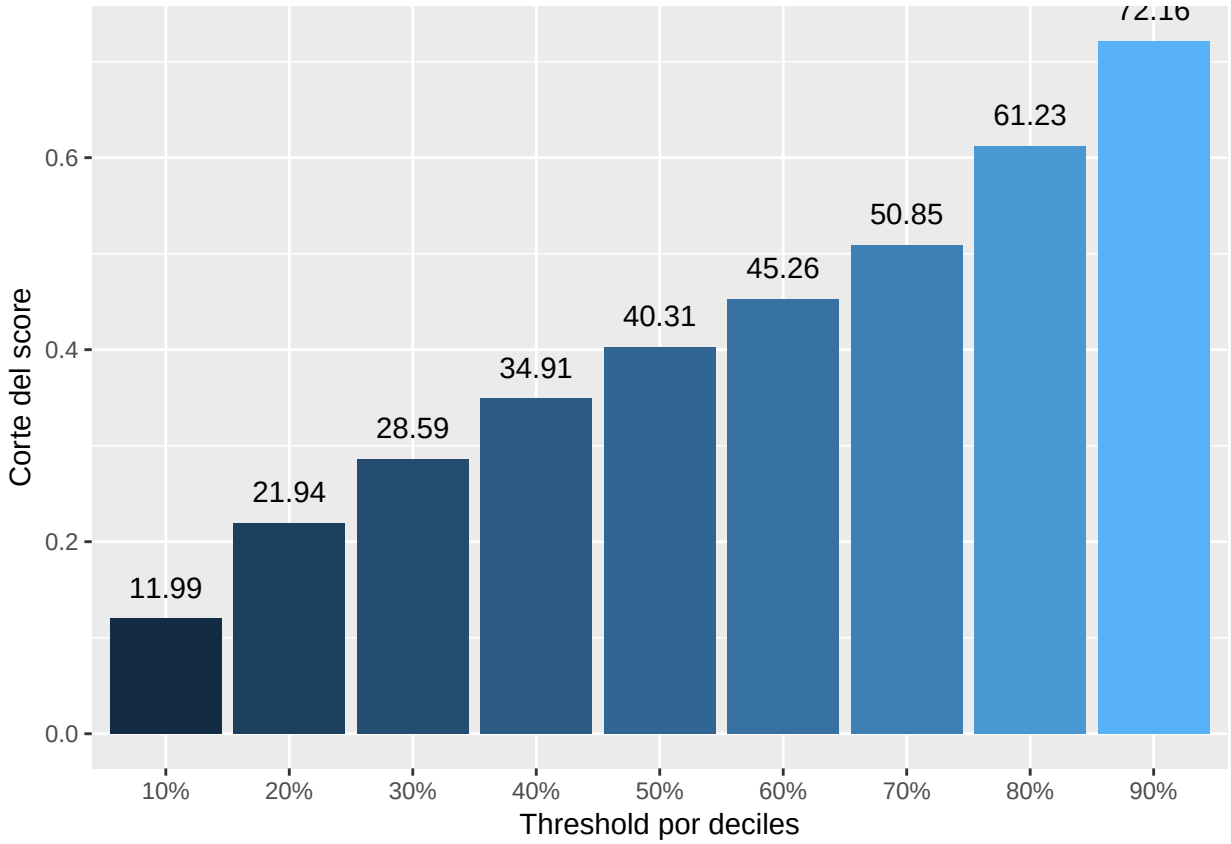
Gráficas para visualizar los resultados

Veamos algunos gráficos:

```
grid.arrange(arrangeGrob(
  ggplot(mat[1:20,1:2],
         aes(x=reorder(Feature,Gain),
             y=Gain,
             label=round(Gain,2),fill=as.numeric(Gain))) +
    geom_col() + coord_flip() + xlab('') + ylab('Importancia') +
    guides(fill=FALSE) + geom_text(hjust=-0.5)))
```

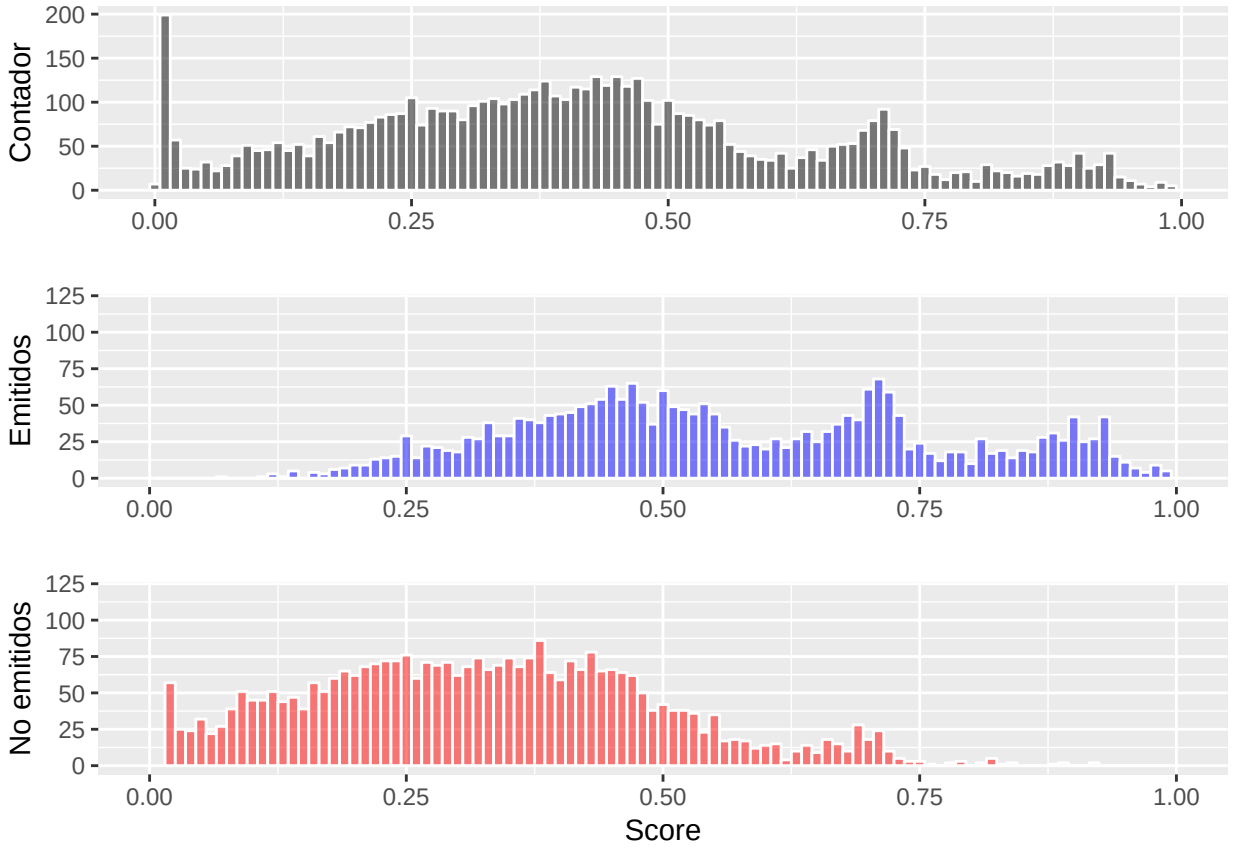



```
ggplot(deciles, aes(
  x=Deciles,
  y=deciles,
  label=round(deciles*100,2),fill=as.numeric(deciles))) +
  geom_col() +
  xlab('Threshold por deciles') + ylab('Corte del score') +
  guides(fill=FALSE) + geom_text(vjust=-1)
```

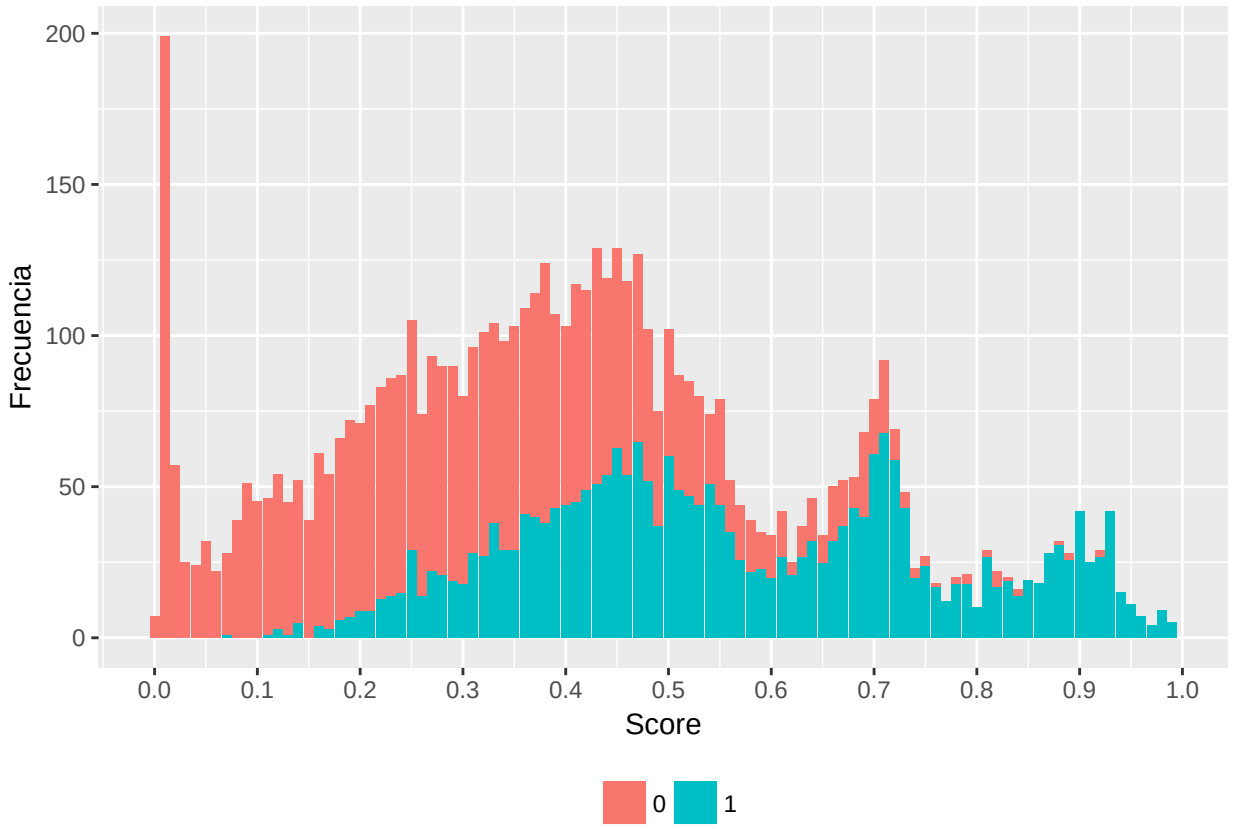



```
grid.arrange(ggplot(as.data.frame(result))+
               geom_histogram(
                 aes(x=score),
                 binwidth = 0.01, fill="black", color="white", alpha=0.5) +
               ylab("Contador") + xlab(''),
             ggplot(filter(result,real==1)) +
               geom_histogram(
                 aes(x=score),
                 binwidth = 0.01, fill="blue", color="white", alpha=0.5) +
               xlab('') + ylab('Emitidos') +
               xlim(0, 1) + ylim(0,120),
             ggplot(filter(result,real==0)) +
               geom_histogram(
                 aes(x=score),
                 binwidth = 0.01, fill="red", color="white", alpha=0.5) +
               xlab('Score') + ylab('No emitidos') +
               xlim(0, 1) + ylim(0,120), ncol=1)
```

```
## Warning: Removed 1 rows containing missing values (geom_bar).
```




```
ggplot(select(result,real,round(score,2)) %>%
         group_by(Score=round(score,2),Emisión=real) %>% tally()) +
  geom_bar(
    aes(x=Score,y=n,fill = as.factor(Emisión)),stat="identity") +
  ylab('Frecuencia') +
  scale_x_continuous(breaks = round(seq(0, 1, by = 0.1),1)) +
  theme(legend.position="bottom", legend.direction="horizontal", legend.title = element_blank())
```




```
xgbi <- xgb.train(params = params,data = dtrain,
  nrounds = 200, #default=bestn para no hacer overfitting
  watchlist = list(val=dtest,train=dtrain),print_every_n = 1,maximize = F,eval_metric = "error")
```

```
## [1]  val-error:0.331643  train-error:0.321464
## [2]  val-error:0.330862  train-error:0.321632
## [3]  val-error:0.326961  train-error:0.315446
## [4]  val-error:0.323449  train-error:0.315446
## [5]  val-error:0.327351  train-error:0.318623
## [6]  val-error:0.321108  train-error:0.315112
## [7]  val-error:0.321888  train-error:0.314611
## [8]  val-error:0.322669  train-error:0.314611
## [9]  val-error:0.317987  train-error:0.308592
## [10] val-error:0.317987  train-error:0.308091
## [11] val-error:0.317206  train-error:0.305751
## [12] val-error:0.314865  train-error:0.304915
## [13] val-error:0.314475  train-error:0.304748
## [14] val-error:0.313695  train-error:0.304246
## [15] val-error:0.314085  train-error:0.301237
## [16] val-error:0.312524  train-error:0.299231
## [17] val-error:0.312524  train-error:0.297392
## [18] val-error:0.313695  train-error:0.296556
## [19] val-error:0.312915  train-error:0.295888
## [20] val-error:0.312524  train-error:0.294885
## [21] val-error:0.312915  train-error:0.293213
## [22] val-error:0.312134  train-error:0.292377
## [23] val-error:0.305892  train-error:0.289034
## [24] val-error:0.305501  train-error:0.288699
```

```
## [25] val-error:0.307452  train-error:0.288198
## [26] val-error:0.309013  train-error:0.288031
## [27] val-error:0.303941  train-error:0.284019
## [28] val-error:0.303160  train-error:0.284186
## [29] val-error:0.302380  train-error:0.283852
## [30] val-error:0.298088  train-error:0.283016
## [31] val-error:0.297698  train-error:0.282180
## [32] val-error:0.298869  train-error:0.280843
## [33] val-error:0.298088  train-error:0.280508
## [34] val-error:0.297308  train-error:0.279338
## [35] val-error:0.295357  train-error:0.279505
## [36] val-error:0.295357  train-error:0.279171
## [37] val-error:0.295357  train-error:0.278168
## [38] val-error:0.291455  train-error:0.276329
## [39] val-error:0.293796  train-error:0.275326
## [40] val-error:0.293796  train-error:0.275159
## [41] val-error:0.293406  train-error:0.274992
## [42] val-error:0.295357  train-error:0.274824
## [43] val-error:0.295357  train-error:0.274156
## [44] val-error:0.295357  train-error:0.273654
## [45] val-error:0.297308  train-error:0.274323
## [46] val-error:0.295357  train-error:0.272150
## [47] val-error:0.295747  train-error:0.272484
## [48] val-error:0.294577  train-error:0.272484
## [49] val-error:0.295357  train-error:0.271481
## [50] val-error:0.294967  train-error:0.272150
## [51] val-error:0.297308  train-error:0.272150
## [52] val-error:0.297308  train-error:0.269141
## [53] val-error:0.298088  train-error:0.268472
## [54] val-error:0.296528  train-error:0.267971
## [55] val-error:0.297698  train-error:0.268472
## [56] val-error:0.297698  train-error:0.266800
## [57] val-error:0.296528  train-error:0.267469
## [58] val-error:0.296918  train-error:0.265965
## [59] val-error:0.296918  train-error:0.266132
## [60] val-error:0.298088  train-error:0.265965
## [61] val-error:0.297698  train-error:0.266633
## [62] val-error:0.296918  train-error:0.266466
## [63] val-error:0.296918  train-error:0.266466
## [64] val-error:0.298869  train-error:0.266132
## [65] val-error:0.298088  train-error:0.265296
## [66] val-error:0.297698  train-error:0.264627
## [67] val-error:0.297698  train-error:0.263791
## [68] val-error:0.296918  train-error:0.263290
## [69] val-error:0.297308  train-error:0.263123
## [70] val-error:0.297698  train-error:0.262120
## [71] val-error:0.298088  train-error:0.261618
## [72] val-error:0.298088  train-error:0.261284
## [73] val-error:0.300819  train-error:0.259278
## [74] val-error:0.300819  train-error:0.259278
## [75] val-error:0.300429  train-error:0.259779
## [76] val-error:0.299649  train-error:0.258609
## [77] val-error:0.299259  train-error:0.257272
## [78] val-error:0.298869  train-error:0.256937
```

```
## [79] val-error:0.298869  train-error:0.256603
## [80] val-error:0.299259  train-error:0.256603
## [81] val-error:0.299649  train-error:0.255934
## [82] val-error:0.300039  train-error:0.255266
## [83] val-error:0.300039  train-error:0.254764
## [84] val-error:0.298869  train-error:0.251421
## [85] val-error:0.298088  train-error:0.251588
## [86] val-error:0.298088  train-error:0.249916
## [87] val-error:0.298478  train-error:0.250084
## [88] val-error:0.298478  train-error:0.250084
## [89] val-error:0.298869  train-error:0.249749
## [90] val-error:0.298478  train-error:0.249081
## [91] val-error:0.298088  train-error:0.249081
## [92] val-error:0.297698  train-error:0.248078
## [93] val-error:0.296918  train-error:0.247576
## [94] val-error:0.297308  train-error:0.248245
## [95] val-error:0.296918  train-error:0.247910
## [96] val-error:0.297698  train-error:0.248078
## [97] val-error:0.298088  train-error:0.247576
## [98] val-error:0.297698  train-error:0.247576
## [99] val-error:0.294967  train-error:0.247409
## [100]    val-error:0.294187  train-error:0.245570
## [101]    val-error:0.295357  train-error:0.246072
## [102]    val-error:0.295357  train-error:0.246072
## [103]    val-error:0.295357  train-error:0.246072
## [104]    val-error:0.297698  train-error:0.245403
## [105]    val-error:0.298869  train-error:0.245403
## [106]    val-error:0.298088  train-error:0.244066
## [107]    val-error:0.297698  train-error:0.241391
## [108]    val-error:0.297308  train-error:0.240889
## [109]    val-error:0.297698  train-error:0.241391
## [110]    val-error:0.296137  train-error:0.239886
## [111]    val-error:0.295357  train-error:0.239886
## [112]    val-error:0.295357  train-error:0.239719
## [113]    val-error:0.296137  train-error:0.239719
## [114]    val-error:0.296528  train-error:0.239050
## [115]    val-error:0.295747  train-error:0.237044
## [116]    val-error:0.295747  train-error:0.237880
## [117]    val-error:0.294577  train-error:0.237212
## [118]    val-error:0.295357  train-error:0.237713
## [119]    val-error:0.294967  train-error:0.237044
## [120]    val-error:0.295747  train-error:0.236543
## [121]    val-error:0.296918  train-error:0.236041
## [122]    val-error:0.296137  train-error:0.235540
## [123]    val-error:0.293796  train-error:0.235540
## [124]    val-error:0.293796  train-error:0.234203
## [125]    val-error:0.295357  train-error:0.233200
## [126]    val-error:0.295747  train-error:0.233200
## [127]    val-error:0.295357  train-error:0.232029
## [128]    val-error:0.293406  train-error:0.232865
## [129]    val-error:0.293406  train-error:0.231194
## [130]    val-error:0.294577  train-error:0.231862
## [131]    val-error:0.294967  train-error:0.230525
## [132]    val-error:0.294577  train-error:0.229188
```

```
## [133]    val-error:0.294577  train-error:0.229188
## [134]    val-error:0.294577  train-error:0.229188
## [135]    val-error:0.294577  train-error:0.228686
## [136]    val-error:0.296528  train-error:0.227014
## [137]    val-error:0.298869  train-error:0.225510
## [138]    val-error:0.298478  train-error:0.225510
## [139]    val-error:0.298088  train-error:0.226011
## [140]    val-error:0.298088  train-error:0.226011
## [141]    val-error:0.297308  train-error:0.225510
## [142]    val-error:0.297698  train-error:0.225677
## [143]    val-error:0.298478  train-error:0.226346
## [144]    val-error:0.299649  train-error:0.225677
## [145]    val-error:0.299259  train-error:0.224841
## [146]    val-error:0.298869  train-error:0.223002
## [147]    val-error:0.298088  train-error:0.221331
## [148]    val-error:0.296528  train-error:0.222166
## [149]    val-error:0.296528  train-error:0.219492
## [150]    val-error:0.297308  train-error:0.218823
## [151]    val-error:0.297698  train-error:0.218656
## [152]    val-error:0.298869  train-error:0.218322
## [153]    val-error:0.299259  train-error:0.218154
## [154]    val-error:0.299259  train-error:0.218322
## [155]    val-error:0.298088  train-error:0.217653
## [156]    val-error:0.298478  train-error:0.215981
## [157]    val-error:0.295747  train-error:0.215313
## [158]    val-error:0.295747  train-error:0.215647
## [159]    val-error:0.296528  train-error:0.215145
## [160]    val-error:0.296137  train-error:0.214978
## [161]    val-error:0.296137  train-error:0.215145
## [162]    val-error:0.296137  train-error:0.214811
## [163]    val-error:0.296137  train-error:0.214811
## [164]    val-error:0.296137  train-error:0.214978
## [165]    val-error:0.296137  train-error:0.214477
## [166]    val-error:0.295747  train-error:0.213641
## [167]    val-error:0.295747  train-error:0.212972
## [168]    val-error:0.296137  train-error:0.211635
## [169]    val-error:0.296918  train-error:0.211301
## [170]    val-error:0.296918  train-error:0.210799
## [171]    val-error:0.297308  train-error:0.209796
## [172]    val-error:0.296528  train-error:0.209629
## [173]    val-error:0.296528  train-error:0.209629
## [174]    val-error:0.296528  train-error:0.209796
## [175]    val-error:0.296137  train-error:0.209629
## [176]    val-error:0.296137  train-error:0.209462
## [177]    val-error:0.296528  train-error:0.209629
## [178]    val-error:0.293016  train-error:0.208292
## [179]    val-error:0.293796  train-error:0.207790
## [180]    val-error:0.292236  train-error:0.206954
## [181]    val-error:0.291846  train-error:0.207121
## [182]    val-error:0.292236  train-error:0.206453
## [183]    val-error:0.293016  train-error:0.205784
## [184]    val-error:0.294187  train-error:0.205283
## [185]    val-error:0.293796  train-error:0.205283
## [186]    val-error:0.294577  train-error:0.204948
```

```
## [187]    val-error:0.295357  train-error:0.203277
## [188]    val-error:0.295357  train-error:0.202608
## [189]    val-error:0.293796  train-error:0.202775
## [190]    val-error:0.294967  train-error:0.201438
## [191]    val-error:0.293796  train-error:0.200769
## [192]    val-error:0.294577  train-error:0.201103
## [193]    val-error:0.295747  train-error:0.201605
## [194]    val-error:0.295747  train-error:0.201605
## [195]    val-error:0.295747  train-error:0.201605
## [196]    val-error:0.296137  train-error:0.201605
## [197]    val-error:0.297308  train-error:0.200602
## [198]    val-error:0.298478  train-error:0.199933
## [199]    val-error:0.298478  train-error:0.199766
## [200]    val-error:0.298088  train-error:0.199933
```

```r
val_error <- as.data.frame(xgbi$evaluation_log$val_error)
train_error <- as.data.frame(xgbi$evaluation_log$train_error)
ggplot(val_error, aes(row(val_error))) +
  geom_line(aes(y = abs(val_error),color='red')) +
  geom_line(aes(y = abs(train_error),color='blue')) +
  xlab('Iteraciones') + ylab('Error') +
  ggtitle('Delta Train & Test Error') + guides(colour=FALSE) +
  geom_vline(xintercept=bestn)  #bestn used in model
```

```
## Don't know how to automatically pick scale for object of type data.frame. Defaulting to continuous.
```

Exportación del modelo

Exportemos ahora nuestro modelo en formato binario para luego ser implementado en nuestro CRM usando Python y XGBoost.

```
xgb.save(xgb1, fname="xgb1.model")
```

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

```
# Chequeo si se exportó bien:
pred <- predict(xgb1,dtrain)
# Cargamos el modelo binario
xgb2 <- xgb.load("xgb1.model")
pred2 <- predict(xgb2, dtrain, ntreelimit = bestn)
# pred2 = pred ? Perfecto:
sum(abs(pred2-pred))
```

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