# Santander

*Pablo Portillo Garrigues*

## Contents



## Introduction

In this paper will perform different approaches to the Santander's Kaggle competition.

The data set consist of 200 features and its a binary classification problem on which the target (outcome) variable represents wheter or not a client makes an specific transaction.

## Auxiliar functions

We will use many auxiliar function on this paper, you can check them out on my github.

```
source("auxFunctions.R")
```

We will define a basic XGB prediction function whose objective will be the logistic one and its metric the AUC.

```
predXGBCM<-function(x,y,test,seed,paramList=c(0.3,1000,5,0,0.75,0.75),model=""){

  suppressMessages(library(xgboost))

  set.seed(seed)



  xgb_params <- list("objective" = "binary:logistic",
```

```
                    "eval_metric" = "auc",

                    eta=paramList[1], gamma=paramList[4], max_depth=paramList[3], subsample=paramList[5

  new_tr <- model.matrix(~.+0,data = x)

  new_ts <- model.matrix(~.+0,data = test)



  dtrain <- xgb.DMatrix(data = new_tr,label = y)



  dtest <- xgb.DMatrix(data = new_ts,label = sample(y,nrow(test),replace = TRUE))



  if(model==""){
   #GPU
    xgb1 <- suppressMessages(xgb.train(params= xgb_params, data = dtrain, nrounds = paramList[2], print_
   #CPU
    #xgb1 <- xgb.train(params= xgb_params, data = dtrain, nrounds = paramList[2], print.every.n = 10)



  }

  else{

    xgb1 <- model



  }


  preds<-predict(xgb1,dtest)

  preds

}
```

## Read the Data

```
setwd("/home/papis/Escritorio/Kaggle/Santander/SeleccionGenetica")
train<-fread("../train.csv",data.table = FALSE)
test<-fread("../test.csv",data.table = FALSE)
```

## A brief exploratory Analysis

```
colnames(train)[which(!(colnames(train) %in%  colnames(test)))]
```

```
## [1] "target"
```

```
which(lapply(train,function(x){sum(is.na(x))})!=0)
```

```
## named integer(0)
```

```
which(lapply(test,function(x){sum(is.na(x))})!=0)
```

```
## named integer(0)
```

```
which(lapply(train,function(x){is.numeric(x)})==0)
```

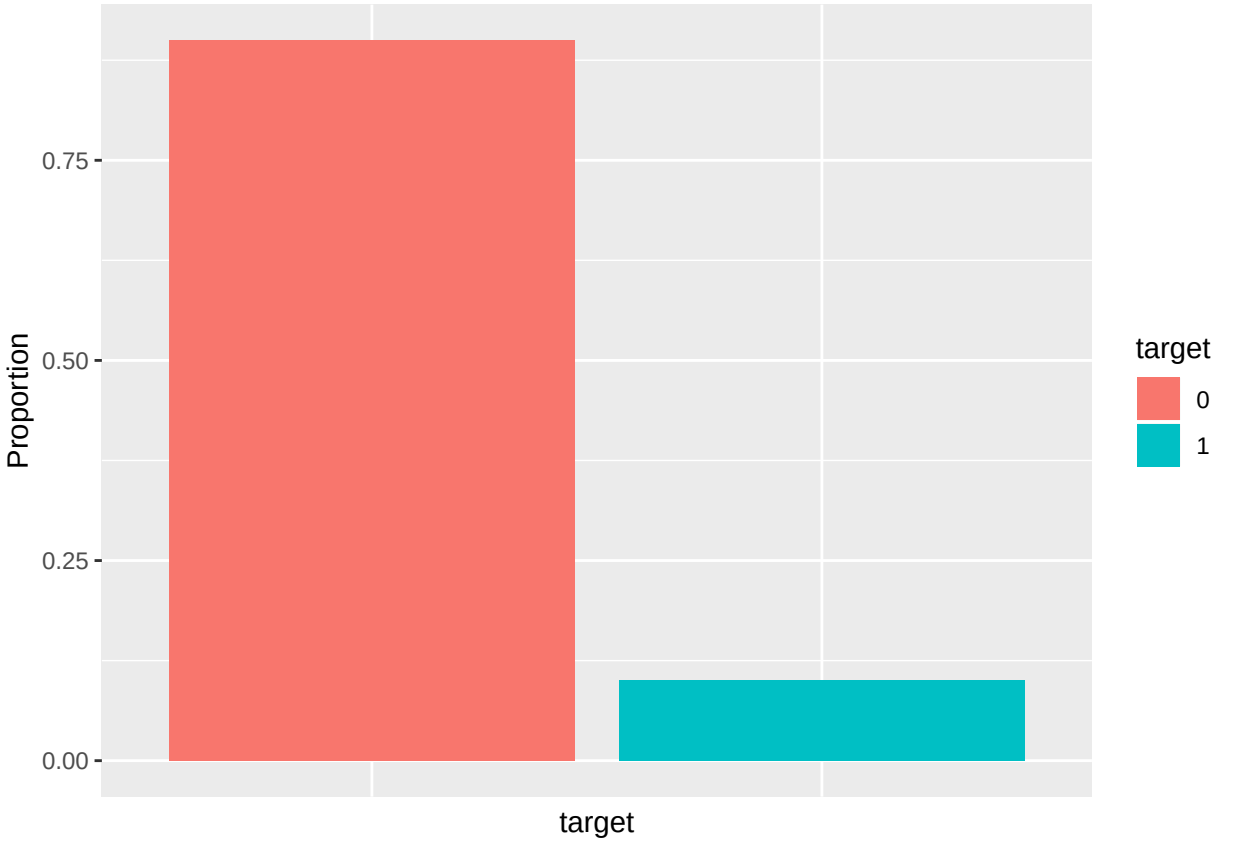
```
## ID_code
##       1
```

```
which(lapply(test,function(x){is.numeric(x)})==0)
```

```
## ID_code
##       1
```

```
train$target<-as.factor(as.character(train$target))


ggplot(train, aes(target)) +
  geom_bar(aes(y=..count../sum(..count..),fill=target)) +
  theme(axis.text.x=element_blank(),
        axis.ticks.x=element_blank())+ labs(y = "Proportion")
```




As we can see there are no differences between the train and the test sets, there are also no missing values and al the variables are numeric.

We can also see that there is an imbalance between the 2 classes of about 9 to 1.

### Variable importance throught Random Forest

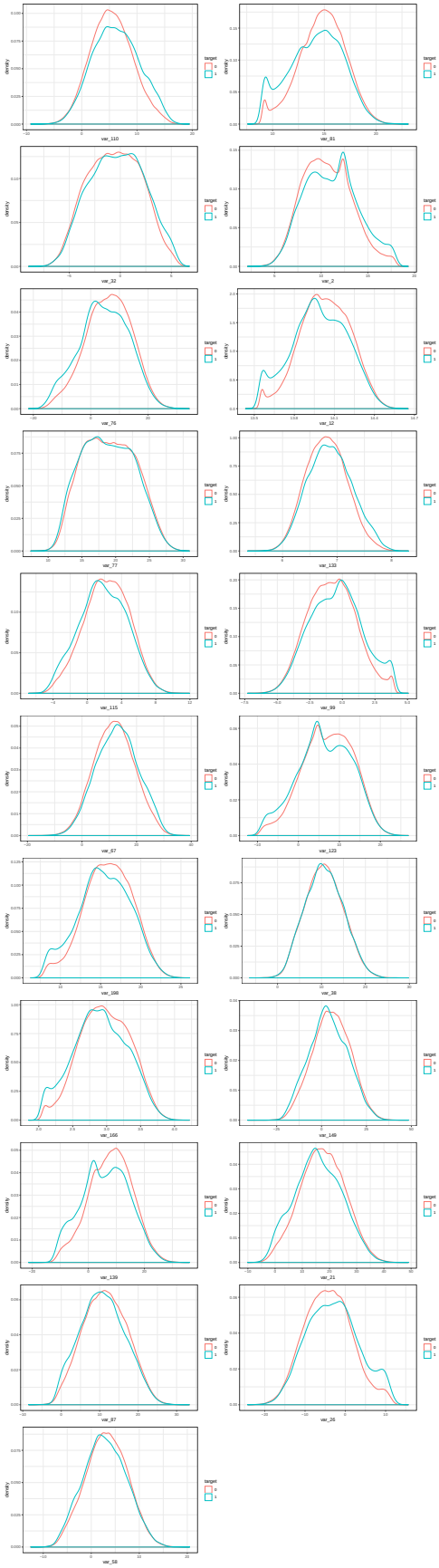
```
#vars<-importRF(train[,3:ncol(train)],train$target)
#vars<-vars[1:20,1]
vars<-c("var_110","var_81","var_32","var_2","var_76","var_12","var_77","var_133","var_115","var_99","va
```

### Plots of the most important variable

We will plot the 20 mots important variables (according to the random forest importance function) discriminating by the target variable to see if there is any difference.

```
plotList<-list()
for(i in vars){
  cont<-which(i==vars)
 p<-ggplot(data = train, aes_string(x = i   , colour = "target",group="target")) +      geom_density(alp
     theme_bw()
    plotList[[cont]] = p
```

```
}
final_plot <- grid.arrange(grobs=plotList, ncol = 2)
```


density
target
0
1
var_110
var_81
var_32
var_2
var_76
var_12
var_77
var_133
var_115
var_99
var_67
var_123
var_198
var_38
var_166
var_148
var_139
var_21
var_87
var_26
var_58

There is hardly any differences between the classes in the most important variables.

### See if there ir correlation between variables

```
correlated<-corrMany(train[,-c(1,2)],0.1)
any(correlated$corr!=0)
```

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

There are no variables with more than 0.1 correlation.

### Data augmentation

Due to the imbalance problem, we are going to augment the data so that we increase the ratio of class 1 to class 0.

We are going to use the 2 leaves kernel Kaggle

```
import pandas as pd
import numpy as np

def augment(train,num_n=2,num_p=1):
    newtrain=[train]
    n=train[train.target==1]
    for i in range(num_n):
        newtrain.append(n.apply(lambda x:x.values.take(np.random.permutation(len(n)))))
    for i in range(num_p):
        p=train[train.target==0]
        newtrain.append(p.apply(lambda x:x.values.take(np.random.permutation(len(p)))))
    return pd.concat(newtrain)
train=pd.read_csv("augmentMe.csv")
t=augment(train)
t.to_csv("trainAug.csv",index=False)
```

```
set.seed(948)
trainAug<-fread("trainAug.csv",data.table =FALSE)
trainAug<-cbind(as.data.frame(paste("train",sample(1:nrow(trainAug)),sep = "_")),trainAug)
colnames(trainAug)[1]<-"ID_code"
```

## 3 different Models

We are going to take 4 different aproaches:

- Apply the ACO to the data with all the variables
- Apply the GAFS and the ACO
- Use the top importance permutation and apply the ACO.

Here you can see more information on the ACO algorithm and you can check the ACO and GAFS algorithm on my Github.

## Apply the ACO algorithm to all the variables

We will first apply the ACO on al the variables to see what parameters for the XGB it suggest us to use.

```
costXGBAco<-function(datos,paramList){
set.seed(123)
cv<-5
suppressMessages(library(dplyr))
  suppressMessages(library(MLmetrics))
  print(Sys.time())
  print((paramList))
  aux<-cvDat(datos[1:200000,],id="ID_code",cv=cv)
  fitness<-c()
  for(i in 1:cv){
    cvID<-which(aux$cvId==i)
    train<-datos[-cvID,]
    test<-datos[cvID,]
    train.y<-train$target
    test.y<-test$target
    train<-train[,!(colnames(train) %in% c("target","ID_code"))]
    test<-test[,!(colnames(test) %in% c("target","ID_code"))]
    preds<-suppressMessages(predXGBCM(x = train,y = train.y,test = test,seed = 123,paramList=paramList))
    fitness<-c(fitness,AUC(preds,test.y))
  }
  print(1-mean(fitness))
  1-mean(fitness)
}


param<-data.frame(eta=c(0.01,0.4),nrounds=c(802,3000),maxDepth=c(2,7),gamma=c(0,0.2),subsample=c(0.5,1)
tip<-c("num","int","int","num","num","num")
#ACO(trainAug,costF=costXGBAco,hor=100,gen=24,tip=tip,paramListR=param)
```

```
costXGBAco(trainAug,paramList=c(0.221456169967382,3100,2,0.2,0.955533727579663,0.335802916466976))
```

```
## [1] "2019-05-15 19:07:23 CEST"
## [1]    0.2214562 3100.0000000    2.0000000    0.2000000    0.9555337
## [6]    0.3358029
## [1] 0.093249
```

```
## [1] 0.093249
```

Our local results are of 0.906751, we have obtain a private score of 0.89567.

## Apply the GAFS and the ACO

We wil use the GAFS algorithm to see what features seems to be the most important. In order to perform the GAFS we are going to define a custome cost function which will consist on a 5-cross validated model of XGB with parameters prefixed. The validation sets will only have rows of the original data (it wont have synthetic rows).

```
costXGBGafs<-function(x,y,cv,seed,datos){
set.seed(seed)
suppressMessages(library(dplyr))
  suppressMessages(library(MLmetrics))
  print(Sys.time())
  print(dim(x))
  aux<-cvDat(datos[1:nrow(x),],id="ID_code",cv=cv)
  fitness<-c()
  for(i in 1:cv){
    cvID<-which(aux$cvId==i)
    train<-datos[-cvID,]
    test<-datos[cvID,]
    train.y<-train$target
    test.y<-test$target
    train<-train[,colnames(train) %in% colnames(x)]
    test<-test[,colnames(test) %in% colnames(x)]
    preds<-suppressMessages(predXGBCM(x = train,y = train.y,test = test,seed = seed ))
    fitness<-c(fitness,AUC(preds,test.y))
  }
  print(mean(fitness))
  mean(fitness)
}

y<-train$target

#selecionar_predictores(train[,3:ncol(train)],y,n_poblacion=100,n_generaciones=24,n_max_predictores=nco
```

As we can see during the execution of the GAFS, it seems that all variables are very usefull for the problem but we will keep the recommendation of using only 195 variables out of the 200 available.

We will now use the ACO alogirhtm over the variables suggested by the GAFS algorithm

```
a<-readLines(con = "genGafs.txt")
vars<-a[length(a)-1]
vars<-sub(pattern = "El mejor individuo desde fuera es:  ",x = vars,replacement = "")
vars<-str_split(vars,pattern = " ")
vars<-unlist(vars)
vars<-vars[-length(vars)]
trainAugGafs<-trainAug[,colnames(trainAug) %in% c(vars,"ID_code","target")]
param<-data.frame(eta=c(0.01,0.4),nrounds=c(802,3000),maxDepth=c(2,7),gamma=c(0,0.2),subsample=c(0.5,1)
tip<-c("num","int","int","num","num","num")
#ACO(trainAugGafs,costF=costXGBAco,hor=100,gen=24,tip=tip,paramListR=param)
```

Using the indications that this process has given us, lets see what CV error it displays

```
costXGBAco(trainAugGafs,paramList=c(0.186756052798322,3000,2,0.164428395167602,0.941608118185663,0.3632
```

```
## [1] "2019-05-15 19:10:17 CEST"
## [1]    0.1867561 3000.0000000    2.0000000    0.1644284    0.9416081
## [6]    0.3632933
## [1] 0.09417153

## [1] 0.09417153
```

Our local results are of 0.9058285, we have obtain a private score of 0.89441.

## XGB importance and ACO

We will use the XGB's importance function and filter out some variables.

```
impXGB<-function(x,y,test,seed,paramList=c(0.3,1000,4,0,1,1)){

  suppressMessages(library(xgboost))
  x<-x[1:200000,]
  y<-y[1:200000]
  set.seed(seed)

  library(xgboost)
  if(is.null(dim(x))){
    x<-as.data.frame(x)
    y<-as.data.frame(y)
  }
  numClass <- length(unique(y))
  params <- list(booster = "gbtree", eta=paramList[1], gamma=paramList[4], max_depth=paramList[3], min_

  new_tr <- model.matrix(~.+0,data = x)

  new_ts <- model.matrix(~.+0,data = test)

  dtrain <- xgb.DMatrix(data = new_tr,label = y)

  dtest <- xgb.DMatrix(data = new_ts,label = sample(y,nrow(test),replace = TRUE))
    #GPU
    xgb1 <- xgb.train(params= params, data = dtrain, nrounds = paramList[2], print_every_n = 10,tree_me
    #CPU
    #xgb1 <- xgb.train(params= params, data = dtrain, nrounds = paramList[2], print.every.n = 10)
    return(xgb.importance(model=xgb1))



}


 imp<-impXGB(trainAug[,-c(1,2)],trainAug$target,test[,-1],seed=123,paramList=c(0.221456169967382,3100,2

imp<-as.data.frame(imp)
imp<-imp[imp$Gain>0.002,]
vars<-c("ID_code","target",imp$Feature)
trainAugImp<-trainAug[,colnames(trainAug) %in% vars]
```

Once we have chosen what threshold to use to choose what variables we are going to keep in the model we will use again the ACO algorithm to select the most appropiate parameters

```
#ACO(trainAugImp,costF=costXGBAco,hor=100,gen=6,tip=tip,paramListR=param)
```

Using the indications that this process has given us, lets see what CV error it displays

```
costXGBAco(trainAugImp,paramList=c(0.201875536463508,3060,2,0.143041371659679,0.845436285351391,0.25))
```

```
## [1] "2019-05-15 19:13:37 CEST"
## [1]    0.2018755 3060.0000000    2.0000000    0.1430414    0.8454363
## [6]    0.2500000
## [1] 0.09511797
```

```
## [1] 0.09511797
```

Our local results are of 0.904882, we have obtain a private score of 0.89404.