CareerCon

Pablo Portillo Garrigues

Contents

Introduction

This is a Kaggle competition where the data set was the measurements that a robot had made (such as its orientation or velocity) and relying on this data the participants had to build models to predict on what surface was the robot standing on.

In this paper I will display different approaches. In order to execute this I use some other functions that you can find on my github.

ReadMe Up!

Not very surprisingly the first thing we are going to do is read the data.

The training data is split in 2 sets: One contains the information of the surface, the series_id and group_id which each measure corresponds to; the other one contains the differents measures each rows has.

```
setwd("/home/papis/Escritorio/Kaggle/careerCon/limpioCareerCon")
trainY<-fread("../y_train.csv",data.table = FALSE)
trainX<-fread("../X_train.csv",data.table = FALSE)
test<-fread("../X_test.csv",data.table = FALSE)
```

We will use an script with auxliar functions available on my github.

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

And we will define a XGB prediction funcion for this scneario.

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

  suppressMessages(library(xgboost))

  set.seed(seed)
```

```
numClass <- length(unique(y))

xgb_params <- list("objective" = "multi:softmax",


                   "eval_metric" = "merror",


                   "num_class" = numClass,

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


  preds<-predict(xgb1,dtest)

  preds

}

else{

  #GPU
  #xgb1 <- xgb.train(params= xgb_params, data = dtrain, nrounds = paramList[2], print.every.n = 10,tr
  #CPU
  xgb1 <- xgb.train(params= xgb_params, data = dtrain, nrounds = paramList[2], print.every.n = 10)
  preds<-predict(xgb1,dtest)
  return(list(preds,xgb.importance(model=xgb1)))
```

```
  }



}
```

Exploratory Analysis

We will seek for any Missing Value, look the different data types the set has and

```
str(trainX)
```

```
## 'data.frame':    487680 obs. of  13 variables:
##  $ row_id               : chr  "0_0" "0_1" "0_2" "0_3" ...
##  $ series_id            : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ measurement_number   : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ orientation_X        : num  -0.759 -0.759 -0.759 -0.759 -0.759 ...
##  $ orientation_Y        : num  -0.634 -0.634 -0.634 -0.634 -0.634 ...
##  $ orientation_Z        : num  -0.105 -0.105 -0.105 -0.105 -0.105 ...
##  $ orientation_W        : num  -0.106 -0.106 -0.106 -0.106 -0.106 ...
##  $ angular_velocity_X   : num  0.10765 0.06785 0.00727 -0.01305 0.00513 ...
##  $ angular_velocity_Y   : num  0.01756 0.02994 0.02893 0.01945 0.00765 ...
##  $ angular_velocity_Z   : num  0.000767 0.003386 -0.005978 -0.008974 0.005245 ...
##  $ linear_acceleration_X: num  -0.749 0.34 -0.264 0.427 -0.51 ...
##  $ linear_acceleration_Y: num  2.1 1.51 1.59 1.1 1.47 ...
##  $ linear_acceleration_Z: num  -9.75 -9.41 -8.73 -10.1 -10.44 ...
```

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

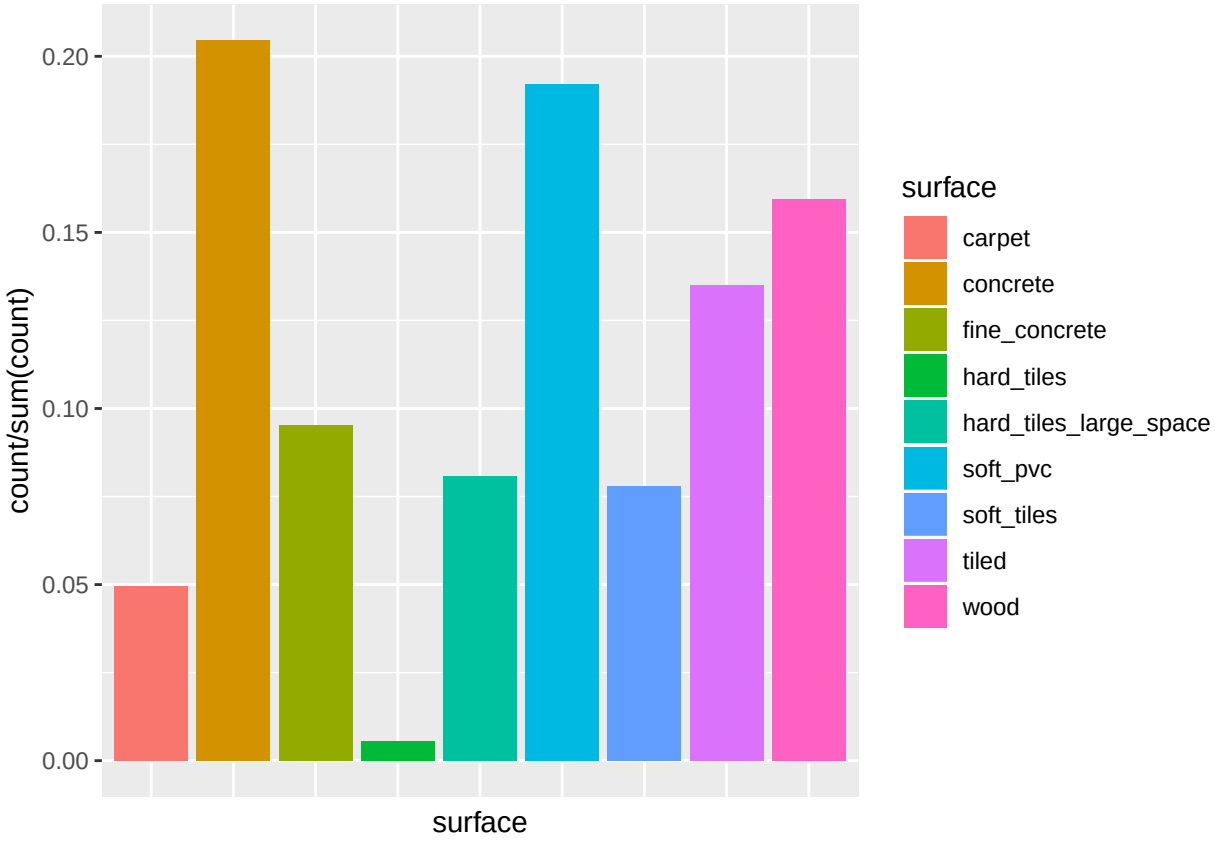
```
## named integer(0)
```

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

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

```
trainY$surface<-as.factor(as.character(trainY$surface))


ggplot(trainY, aes(surface,fill=surface)) +
  geom_bar(aes(y=..count../sum(..count..))) + theme(axis.text.x = element_blank(),axis.ticks.x=element_
```



As we can see, we have a clear problem oc calss imbalance. This wouldnt be a problem if the test data follow a similar distribution, but we have observed that this is not the case.

Feature Engineering

So, as we have seen the training set has multiple rows for the same id, we might be wondering id we should do something about it.

We will need to somehow aggregate the data by the id because we cant have dependent rows within the training set.

We wil create some features based on the aggreagtions of row with equal ID.

The different orientation variables are expressed as quaternions so we will proceed to transform them to degrees

```
quaternion_to_euler<-function(x,y,z,w){

    t0 <- +2.0 * (w * x + y * z)
    t1<- 1.0 - 2.0 * (x * x + y * y)
    X<-atan2(t0, t1)

    t2<-2.0 * (w * y - z * x)
    if(t2>1){
      1
    }
    else{
```

```
      t2
    }
    if(t2<-1){
      -1
    }
    else{
      t2
    }
    Y<-asin(t2)

    t3<-2.0 * (w * z + x * y)
    t4<-1.0 - 2.0 * (y * y + z * z)
    Z<-atan2(t3, t4)
    c(X,Y,Z)

}
```

We will create new features which will be a normalized version of the orientations.

```
step0<-function(actual){
  actual[,'norm_quat'] = (actual[,'orientation_X']**2 +            actual[,'orientation_Y']**2 + actual
  actual[,'mod_quat'] = (actual[,'norm_quat'])**0.5
  actual[,'norm_X'] = actual[,'orientation_X'] / actual[,'mod_quat']
  actual[,'norm_Y'] = actual[,'orientation_Y'] / actual[,'mod_quat']
  actual[,'norm_Z'] = actual[,'orientation_Z'] / actual[,'mod_quat']
  actual[,'norm_W'] = actual[,'orientation_W'] / actual[,'mod_quat']
  actual
}
```

This function will use the quaternion_to_euler to transform the normalized versions of the quaternions_orientation to degrees.

```
step1<-function(actual){
    x<-actual[,'norm_X']
    y<-actual[,'norm_Y']
    z<-actual[,'norm_Z']
    w<-actual[,'norm_W']
    nx<-c()
    ny<-c()
    nz<-c()
    for(i in 1:length(x)){
        aux <- quaternion_to_euler(x[i], y[i], z[i], w[i])
        nx<-c(nx,aux[1])
        ny<-c(ny,aux[1])
        nz<-c(nz,aux[1])
    }
    actual[,'euler_x'] = nx
    actual[,'euler_y'] = ny
    actual[,'euler_z'] = nz
    actual
}
```

We will proceed to transform the quaternions to degrees. We will also add new variables, total angular velocity, total linear accelaration and total orientation.

Then we will merge the trainX and trainY so that we can create new features.

```
trainX<-step0(trainX)
trainX<-step1(trainX)

trainX$totl_anglr_vel<-(trainX['angular_velocity_X']^2 + trainX['angular_velocity_Y']^2 + trainX['ang

trainX$totl_linr_acc<-(trainX['linear_acceleration_X']^2 + trainX['linear_acceleration_Y']^2 + trainX

trainX$totl_xyz<-(trainX['orientation_X']^2 + trainX['orientation_Y']^2 + trainX['orientation_Z']^2)


train<-merge(x=trainX,y=trainY,by.x="series_id",by.y="series_id",all = TRUE)
```

We will add some basic aggregative features such as the mean, standard deviation or the median.

```
vars<-c("range","min","max","25th","_mean_change_of_abs_change","75th","maxToMin","mean","median","sd",

extVars<-lapply(4:(ncol(train)-2),function(x){
    df<-as.data.frame(replicate(length(vars), rnorm(length(unique(train$series_id)))))
    colnames(df)<-paste(colnames(train)[x],vars,sep="_")
    df$series_id<-rnorm(length(unique(train$series_id)))
    for(i in 1:length(unique(train$series_id))){
        ser<-unique(train$series_id)[i]
        aux<-train[train$series_id==ser,]
        df[i,1]<-abs(max(aux[,x])-min(aux[,x]))
        df[i,2]<-min(aux[,x])
        df[i,3]<-max(aux[,x])
        df[i,4]<-quantile(aux[,x],0.25)
        df[i,5]<-mean(diff(abs(diff(aux[,x]))))
        df[i,6]<-quantile(aux[,x],0.75)
        df[i,7]<-max(aux[,x])/min(aux[,x])
        df[i,8]<-mean(aux[,x])
        df[i,9]<-median(aux[,x])
        df[i,10]<-sd(aux[,x])
        df[i,11]<-mean(abs(diff(aux[,x])))
        df[i,12]<-min(abs(aux[,x]))
        df[i,13]<-max(abs(aux[,x]))
        df[i,14]<-(df[i,12]+df[i,13])/2
        df[i,15]<-ser
    }
    df

})



extM<-extVars[[1]][1:3810,]
for(i in 2:length(extVars)){
aux<-extVars[[i]][1:3810,]
extM<-merge(x=extM,y=aux,by.x="series_id",by.y="series_id",all = TRUE)

}
```

```
extM<-merge(x=extM,y=trainY,by.x="series_id",by.y="series_id",all = TRUE)
```

Miscellaneous plots

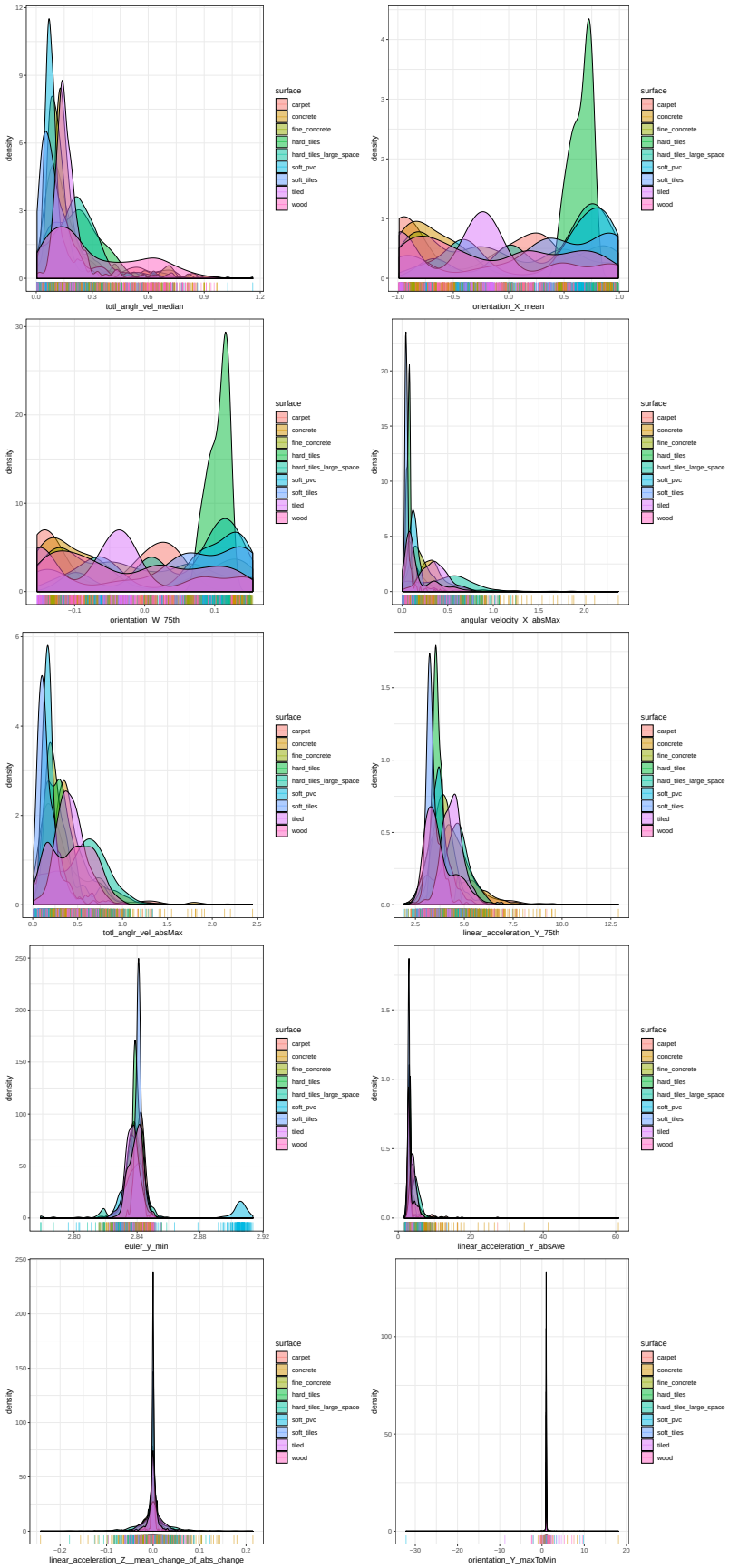
We will plot the density of some variables to see if the different surface have different densities for these variables.

```
plotList<-list()
vars<-c("totl_anglr_vel_median","orientation_X_mean","orientation_W_75th","angular_velocity_X_absMax","
for(i in vars){
  cont<-which(i==vars)

 p<-ggplot(extM, aes_string(x = i   , fill="surface")) + geom_density(alpha = 0.5)+
      geom_rug(aes(color = surface), alpha = 0.5) +
      theme_bw()
    plotList[[cont]] = p

}


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


density
totl_anglr_vel_median
orientation_X_mean
orientation_W_75th
angular_velocity_X_absMax
totl_anglr_vel_absMax
linear_acceleration_Y_75th
euler_y_min
linear_acceleration_Y_absAve
linear_acceleration_Z__mean_change_of_abs_change
orientation_Y_maxToMin
surface
carpet
concrete
fine_concrete
hard_tiles
hard_tiles_large_space
soft_pvc
soft_tiles
tiled
wood

Some things worth mentioning are the bumps that occur for the concrete surface in the graphs of orientation_X_mean and orientation_W_75th. Also the indistinguishable mixture of surfaces in the variables linear_acceleration_Z___mean_change_of_abs_change and orientation_Y_maxToMin.

You might have notice the incredible resemblance that have the variables orientation_X_mean and orientation_W_75th, this is due to the correlation between many variables. We will deal with this in the last section.

Dealing with class Imbalance

As we have seen, there is a class target imbalance. Having 9 different surfaces we will try to reach the even proportion (100/9). To achieve this goal, we will undersample those overrepresentated surfaces and augment those minority surfaces.

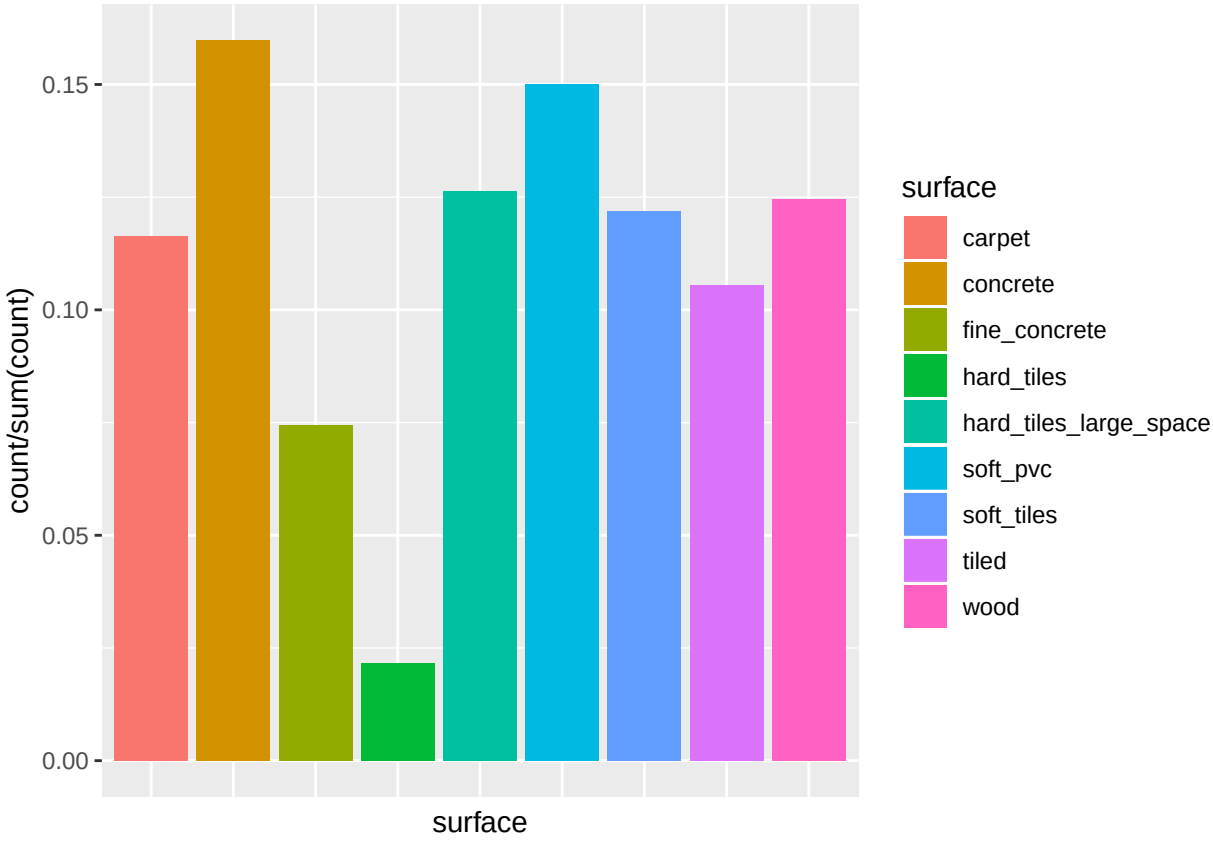
Being more Specific, we will generate synthetic data for the surfaces hard_tiles, wood, soft_tiles and hard_tiles_large_space.

```
set.seed(64920)
trainAug<-Aug(datos = extM[,-c(1,310)])
trainAug$group_id<-c(extM$group_id,rep(6666,(nrow(trainAug)-nrow(extM))))


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

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

```
ggplot(trainAug, aes(surface,fill=surface)) +
  geom_bar(aes(y=..count../sum(..count..))) + theme(axis.text.x = element_blank(),axis.ticks.x=element_
```



We will also undersample the conrete and soft_pvc surfaces. Instead of deleting random series_id we will delete group_ids.

```
auxConc<-as.data.frame(trainY[trainY$surface=="concrete",] %>% group_by(group_id) %>% summarise(seriesN=
auxConc<-auxConc[order(auxConc$seriesN,decreasing = TRUE),]
auxConc
```

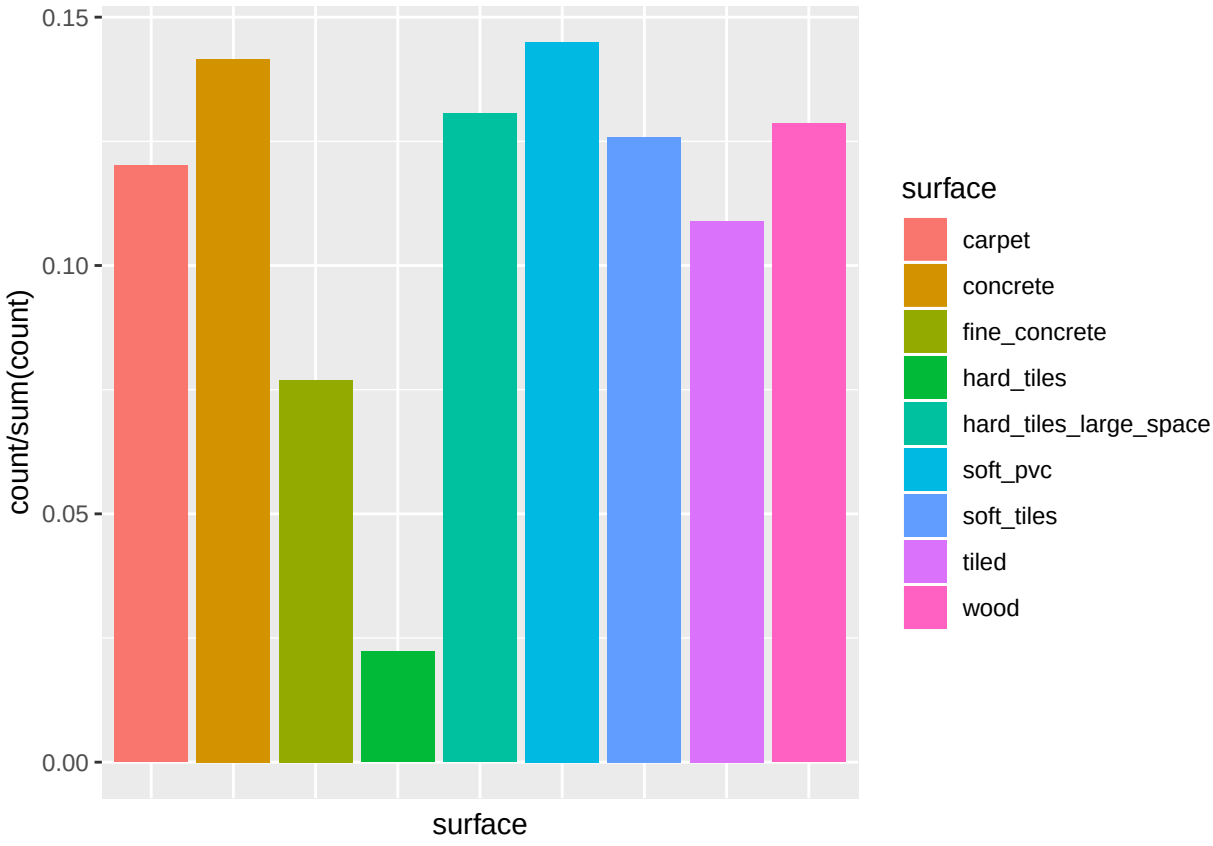
```
##    group_id seriesN
## 7        39      70
## 8        41      70
## 9        42      70
## 10       47      70
## 13       61      70
## 14       62      70
## 1         0      57
## 3        12      57
## 4        20      57
## 6        32      57
## 5        31      54
## 12       57      41
## 15       63      17
## 11       50      10
## 2         5       9
```

```
auxSpvc<-as.data.frame(trainY[trainY$surface=="soft_pvc",] %>% group_by(group_id) %>% summarise(seriesN=
auxSpvc<-auxSpvc[order(auxSpvc$seriesN,decreasing = TRUE),]

auxSpvc
```

```
##    group_id seriesN
## 10       53      71
## 14       70      71
## 13       69      70
## 9        51      60
## 1         3      57
## 3        18      57
## 4        19      57
## 5        26      57
## 6        29      57
## 7        34      57
## 2         6      48
## 8        37      34
## 11       56      29
## 12       58       7
```

```
ggplot(trainAug[!(trainAug$group_id %in% c(39,57,6)),], aes(surface,fill=surface)) +
  geom_bar(aes(y=..count../sum(..count..))) + theme(axis.text.x = element_blank(),axis.ticks.x=element_
```



It seems that now we have a more or less better distributed target.

Dependence between rows of the same group_id

To prove this, lets make a simple test: we will build 2-cross validated sets, one separating between group_ids and the other between series_ids.

```
CVProofGroup<-function(datos,paramList=c(0.3,300,5,0,0.75,0.75)){
  set.seed(240)
  library(dplyr)
  library(MLmetrics)
  vars<-colnames(datos)[1:308]
    colnames(datos)[309]<-"target"
  datos<-cvDat(datos,cv=5,id="group_id")
  fit<-c()
  for(i in 1:5){
    train<-datos[datos$cvId!=i,]
    test<-datos[datos$cvId==i,]
    train.y<-train$target
    test.y<-test$target
    train<-train[,vars]
    test<-test[,vars]
    preds<-suppressWarnings(predXGBCM(x=train,y=charInteger(train.y),test=test,paramList=paramList,seed=
    fit<-c(fit,Accuracy(preds,charInteger(test.y)))
  }
  mean(fit)
}



CVProofSeries<-function(datos,paramList=c(0.3,300,5,0,0.75,0.75)){
  set.seed(240)
  library(dplyr)
  library(MLmetrics)
  vars<-colnames(datos)[1:308]
    colnames(datos)[309]<-"target"
    datos$series_id<-1:nrow(datos)
  datos<-cvDat(datos,cv=5,id="series_id")
  fit<-c()
  for(i in 1:5){
    train<-datos[datos$cvId!=i,]
    test<-datos[datos$cvId==i,]
    train.y<-train$target
    test.y<-test$target
    train<-train[,vars]
    test<-test[,vars]
    preds<-suppressWarnings(predXGBCM(x=train,y=charInteger(train.y),test=test,paramList=paramList,seed=
    fit<-c(fit,Accuracy(preds,charInteger(test.y)))
  }
  mean(fit)
}

CVProofSeries(datos=trainAug)
```

```
##
## Attaching package: 'MLmetrics'
```

```
## The following object is masked from 'package:base':
##
##     Recall
```

```
## [1] 0.9196467
```

```
CVProofGroup(datos=trainAug)
```

```
## [1] 0.7516637
```

As we can see the results are bastly different and the only differences are how are the the validation sets created. We can infer that there is indeed a big dependence within group_id so we will refine our models using a validation set built by hand.

The group_ids that we are going to use for the validation set are: 8, 10, 11, 14, 16, 22, 25, 27, 36, 37, 41, 42, 57, 69, 70

Model Building

You can refer to the different algorithms used in the following section in my Gihub and my rpubs site.

We will benchmark the following approaches:

- Use the Permutation Importance function for Feature Selection and then apply the ACO on an XGB model.
- Use the GAFS for Feature Selection and the ACO on an XGB model.
- Use all the varibles and apply the ACO on an XGB model.
- Use the Permutation Importance function for Feature Selection and then apply the ACO on an Random Forest model.
- Uncorrelated Features and ACO

Permutation, ACO on XGB

```
costPerm<-function(datos,cv,rep,seed,importancia=""){
  set.seed(seed)
  library(dplyr)
  library(MLmetrics)
  vars<-colnames(datos)[1:308]
  test<-datos[(datos$group_id %in% c(8, 10, 11, 14, 16, 22, 25,  27, 36, 37, 41, 42, 57, 69, 70)),]
  train<-datos[!(datos$group_id %in% c(8, 10, 11, 14, 16, 22, 25, 36, 37, 41, 42, 57, 69, 70,39,6)),]
  train.y<-train$surface
  test.y<-test$surface
  train<-train[,vars]
  test<-test[,vars]

  if(importancia==""){
    preds<-suppressWarnings(predXGBCM(x=train,y=charInteger(train.y),test=test,paramList=c(0.3,300,5,0,
    Accuracy(preds,charInteger(test.y))
  }

  else{
```

```
    aux<-predXGBCM(x=train,test=test,y = train.y,seed = seed,importancia=importancia,paramList=c(0.3,300
    return(list(Accuracy(aux[[1]],charInteger(test.y)),aux[[2]]))
  }

}


#importPerm(datos=trainAug,costF=costPerm,cv=1,rep=1,control=1,perm=15,seed=240)

aux<-readLines("iter.txt")
aux<-sub(pattern = "La variable ",aux,replacement = "")
dfVars<-as.data.frame(matrix(unlist(str_split(string = aux," tiene una diferencia de error respecto a la
dfVars$V2<-as.numeric(as.character(dfVars$V2))
dfVars<-dfVars[order(dfVars$V2,decreasing = TRUE),]
```

To give a brief explanation, what the algorithm importPerm does is calculate the Error of a model with all its variables, and then calculates again its error having permutted the value of one variable. If the error has improved even thought the variable has been permutted we can infer that the variable in question is not very useful.

We have chosen a threshold of improvement of 0.5%, there are 37 variables that improve by 0.1% or more after being permutted.

```
vars<-as.character(dfVars[which(dfVars[,2] <= 0.005),1])



antACO<-function(datos,paramList=c(0.3,300,5,0,0.75,0.75)){
  library(dplyr)
  library(MLmetrics)
  library(xgboost)


  test<-datos[(datos$group_id %in% c(8, 10, 11, 14, 16, 22, 25,  27, 36, 37, 41, 42, 57, 69, 70)),]
  train<-datos[!(datos$group_id %in% c(8, 10, 11, 14, 16, 22, 25, 36, 37, 41, 42, 57, 69, 70,39,6)),]

  vars<-colnames(datos)
  print(paramList)
  beg.time<-Sys.time()
  set.seed(240)
  train.y<-train$surface
  test.y<-test$surface
  train<-train[,!(colnames(train) %in% c("surface","group_id"))]
  test<-test[,!(colnames(test) %in% c("surface","group_id"))]
  train.y<-charInteger(train.y)
  test.y<-charInteger(test.y)
  preds<-predXGBCM(x=train,y=train.y,test=test,paramList=paramList,seed=123)
  cat(as.character(beg.time), " Hormiga ejecutada\n", file="horExec.txt",append=TRUE)
  1-Accuracy(preds,test.y)

}
#source("ACOr.R")

data<-trainAug[,colnames(trainAug) %in% c(vars,"surface","group_id")]
```

```
param<-data.frame(eta=c(0.01,0.4),nrounds=c(802,3100),maxDepth=c(2,7),gamma=c(0,0.2),subsample=c(0.5,1)
tip<-c("num","int","int","num","num","num")
#ACO(data,costF=antACO,hor=100,gen=24,tip=tip,paramListR=param)
```

The ACO algorithm suggest the use of the following parameters: 0.0149587062518246, 1081, 4, 0.117388696700266, 0.571154546018041, 0.646760882304712

```
antACO(data,c(0.0149587062518246, 1081, 4, 0.117388696700266, 0.571154546018041, 0.646760882304712))
```

```
## [1] 1.495871e-02 1.081000e+03 4.000000e+00 1.173887e-01 5.711545e-01
## [6] 6.467609e-01
```

```
## Warning: 'print.every.n' is deprecated.
## Use 'print_every_n' instead.
## See help("Deprecated") and help("xgboost-deprecated").
```

```
## [1] 0.3780025
```

Our local results are of 0.6093552, we have obtain a private score of 0.6049.

GAFS, ACO on XGB

```
costGafsXGB<-function(x,y,cv,seed,datos){
  library(dplyr)
  library(MLmetrics)
  library(xgboost)
    set.seed(123)

  test<-datos[(datos$group_id %in% c(8, 10, 11, 14, 16, 22, 25,  27, 36, 37, 41, 42, 57, 69, 70)),]
  train<-datos[!(datos$group_id %in% c(8, 10, 11, 14, 16, 22, 25, 36, 37, 41, 42, 57, 69, 70,39,6)),]

  vars<-colnames(x)
  print(dim(x))
  beg.time<-Sys.time()
  set.seed(240)
  train.y<-train$surface
  test.y<-test$surface
  train<-train[,(colnames(train) %in% vars)]
  test<-test[,(colnames(test) %in% vars)]
  train.y<-charInteger(train.y)
  test.y<-charInteger(test.y)
  preds<-predXGBCM(x=train,y=train.y,test=test,paramList=paramList,seed=123)
  cat(as.character(beg.time), " Hormiga ejecutada\n", file="horExec.txt",append=TRUE)
  1-Accuracy(preds,test.y)

}

x<-trainAug[,!(colnames(trainAug) %in% c("surface","group_id","series_id"))]
y<-trainAug$surface
```

```
#selecionar_predictores(x,y,n_poblacion=2,n_generaciones=24,n_max_predictores=ncol(x),n_min_predictores

a<-readLines(con = "gen.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)]
```

```
#vars<-as.character(dfVars[which(dfVars[,2] <= 0.005),1])


data<-trainAug[,colnames(trainAug) %in% c(vars,"surface","group_id")]

param<-data.frame(eta=c(0.01,0.4),nrounds=c(802,3100),maxDepth=c(2,7),gamma=c(0,0.2),subsample=c(0.5,1)
tip<-c("num","int","int","num","num","num")
#ACO(data,costF=antACO,hor=100,gen=24,tip=tip,paramListR=param)
```

The ACO algorithm suggest the use of the following parameters: 0.378169067181216, 3100, 2, 0.0540014568383044, 0.666640054375634, 0.467106912285765

```
antACO(data,c(0.378169067181216, 3100, 2, 0.0540014568383044, 0.666640054375634, 0.467106912285765))
```

```
## [1] 3.781691e-01 3.100000e+03 2.000000e+00 5.400146e-02 6.666401e-01
## [6] 4.671069e-01
```

```
## Warning: 'print.every.n' is deprecated.
## Use 'print_every_n' instead.
## See help("Deprecated") and help("xgboost-deprecated").
```

```
## [1] 0.3931732
```

Our local results are of 0.6030341, we have obtain a private score of 0.5864.

All the variables, ACO on XGB

```
antACO<-function(datos,paramList=c(0.3,300,5,0,0.75,0.75)){
  library(dplyr)
  library(MLmetrics)
  library(xgboost)


  test<-datos[(datos$group_id %in% c(8, 10, 11, 14, 16, 22, 25,  27, 36, 37, 41, 42, 57, 69, 70)),]
  train<-datos[!(datos$group_id %in% c(8, 10, 11, 14, 16, 22, 25, 36, 37, 41, 42, 57, 69, 70,39,6)),]

  vars<-colnames(datos)
  print(paramList)
  beg.time<-Sys.time()
  set.seed(240)
```

```
  train.y<-train$surface
  test.y<-test$surface
  train<-train[,!(colnames(train) %in% c("surface","group_id"))]
  test<-test[,!(colnames(test) %in% c("surface","group_id"))]
  train.y<-charInteger(train.y)
  test.y<-charInteger(test.y)
  preds<-predXGBCM(x=train,y=train.y,test=test,paramList=paramList,seed=123)
  cat(as.character(beg.time), " Hormiga ejecutada\n", file="horExec.txt",append=TRUE)
  1-Accuracy(preds,test.y)

}
data<-trainAug

param<-data.frame(eta=c(0.01,0.4),nrounds=c(802,3100),maxDepth=c(2,7),gamma=c(0,0.2),subsample=c(0.5,1)
tip<-c("num","int","int","num","num","num")
#ACO(data,costF=antACO,hor=100,gen=24,tip=tip,paramListR=param)
```

The ACO algorithm suggest the use of the following parameters: 0.222837059191604, 2337, 4, 0.113599783871105, 0.715732562851235, 0.38113991626427

```
antACO(data,c( 0.222837059191604, 2337, 4, 0.113599783871105, 0.715732562851235, 0.38113991626427))
```

```
## [1]    0.2228371 2337.0000000    4.0000000    0.1135998    0.7157326
## [6]    0.3811399

## Warning: 'print.every.n' is deprecated.
## Use 'print_every_n' instead.
## See help("Deprecated") and help("xgboost-deprecated").

## [1] 0.3830594
```

Our local results are of 0.6169406, we have obtain a private score of 0.5911.

Permutation, ACO on Random Forest

```
antAcoRF<-function(datos,paramList=c(0,1000,1), seed=123){
  library(dplyr)
  library(MLmetrics)
  source("auxFunctions.R")


  predRFC<-function(x,y,test,paramList=c(0,1000,1),seed){
    if(paramList[1]==0){
      paramList[1]=sqrt(ncol(x))
    }
    set.seed(seed)
    model<-randomForest::randomForest(x=x,y=as.factor(y),mtry=paramList[1],ntree=paramList[2],nodesize=
    predict(model,test)
  }
  test<-datos[(datos$group_id %in% c(8, 10, 11, 14, 16, 22, 25,  27, 36, 37, 41, 42, 57, 69, 70)),]
```

```
  train<-datos[!(datos$group_id %in% c(8, 10, 11, 14, 16, 22, 25, 36, 37, 41, 42, 57, 69, 70,39,6)),]

  vars<-colnames(datos)
  print(paramList)
  beg.time<-Sys.time()
  set.seed(240)
  train.y<-train$surface
  test.y<-test$surface
  train<-train[,!(colnames(train) %in% c("surface","group_id"))]
  test<-test[,!(colnames(test) %in% c("surface","group_id"))]
  preds<-predRFC(x=train,y=train.y,test=test,paramList=paramList,seed=123)
  cat(as.character(beg.time), " Hormiga ejecutada\n", file="horExec.txt",append=TRUE)
  1-Accuracy(preds,test.y)


}


param<-data.frame(mtry=c(1,sqrt(ncol(trainAug))),ntree=c(500,3000),nodesize=c(1,nrow(trainAug)/100))
tip<-c("int","int","int")
#ACO(trainAug,costF=antAcoRF,hor=100,gen=24,tip=tip,paramListR=param)
```

The ACO algorithm suggest the use of the following parameters: 4, 500, 9

```
antAcoRF(data,c( 4, 500, 9))
```

```
## [1]   4 500   9
```

```
## [1] 0.3780025
```

Our local results are of 0.6219975, we have obtain a private score of 0.5793.

Uncorrelated features and ACO

We will first of all delete those features with a correlation higer than 0.8

```
tmp <- cor(extM[,!(colnames(extM) %in% c("series_id","group_id","surface"))])
tmp[upper.tri(tmp)] <- 0
diag(tmp) <- 0
data.new <- extM[,!apply(tmp,2,function(x) any(x > 0.8))]
data.new$surface<-extM$surface

set.seed(64920)
trainAugUncorr<-Aug(datos = data.new)
trainAugUncorr$group_id<-c(extM$group_id,rep(6666,(nrow(trainAug)-nrow(extM))))
```

As we can see, there are only 66 features with a correlation less than 0.8. We will use only those and run the ACO on them.

```
antACO<-function(datos,paramList=c(0.3,300,5,0,0.75,0.75)){
  library(dplyr)
  library(MLmetrics)
  library(xgboost)


  test<-datos[(datos$group_id %in% c(8, 10, 11, 14, 16, 22, 25,  27, 36, 37, 41, 42, 57, 69, 70)),]
  train<-datos[!(datos$group_id %in% c(8, 10, 11, 14, 16, 22, 25, 36, 37, 41, 42, 57, 69, 70,39,6)),]

  vars<-colnames(datos)
  print(paramList)
  beg.time<-Sys.time()
  set.seed(240)
  train.y<-train$surface
  test.y<-test$surface
  train<-train[,!(colnames(train) %in% c("surface","group_id"))]
  test<-test[,!(colnames(test) %in% c("surface","group_id"))]
  train.y<-charInteger(train.y)
  test.y<-charInteger(test.y)
  preds<-predXGBCM(x=train,y=train.y,test=test,paramList=paramList,seed=123)
  cat(as.character(beg.time), " Hormiga ejecutada\n", file="horExec.txt",append=TRUE)
  1-Accuracy(preds,test.y)

}


param<-data.frame(eta=c(0.01,0.4),nrounds=c(802,3100),maxDepth=c(2,7),gamma=c(0,0.2),subsample=c(0.5,1)
tip<-c("num","int","int","num","num","num")
#ACO(trainAugUncorr,costF=antACO,hor=100,gen=24,tip=tip,paramListR=param)
```

Lets take a look at the cv resuslts

```
antACO(trainAugUncorr,c(0.0360144148214974 , 1664, 7,0.0652696915844918,0.620187458168021,0.55877447819
```

```
## [1] 3.601441e-02 1.664000e+03 7.000000e+00 6.526969e-02 6.201875e-01
## [6] 5.587745e-01
```

```
## Warning: 'print.every.n' is deprecated.
## Use 'print_every_n' instead.
## See help("Deprecated") and help("xgboost-deprecated").
```

```
## [1] 0.391909
```

Our local results are of 0.608091, we have obtain a private score of 0.5788