Will it Burn?

Pablo Portillo Garrigues

May 20, 2019

Contents

Introduction

The main goal on this notebook is to predict on real time wheter or not a post of reddit will get to the hot sections of its subreddit.

Reddit is a popular social network with millions of daily users and hundred of thousands of posts. It would bee interesting to develope a tool that allowed us to differentiate in real time between potential succesfull post and forgotten posts. Reddit has an easy to use API called Praw.

Due to the reasons described above we have chosen reddit as the playground for our machine learning activities.

We will also build a bot who will comment on a post that we predict to be a potenital hot post.

Read the data

```
setwd("/home/papis/Escritorio/willItBurn-/scrappingScripts/posts/postFInales/")
temp = list.files(pattern="*.csv")
myfiles = lapply(temp, read.csv)
dat<-myfiles[[1]]
for(i in 2:length(myfiles)){
  dat<-rbind(dat,myfiles[[i]])
}
setwd("/home/papis/Escritorio/willItBurnBotOld/preProcessing/")
```

Exploratory Analysis

```
dim(dat)
```

```
## [1] 661516     16
```

```
colnames(dat)
```

```
##  [1] "title"        "author"       "edited"       "is_self"
##  [5] "locked"       "spoiler"      "stickied"     "score"
##  [9] "id"           "subreddit"    "url"          "num_comments"
## [13] "hot"          "body"         "created"      "measured"
```

```
length(unique(dat$id))
```

```
## [1] 59224
```

```
dat[1,]
```

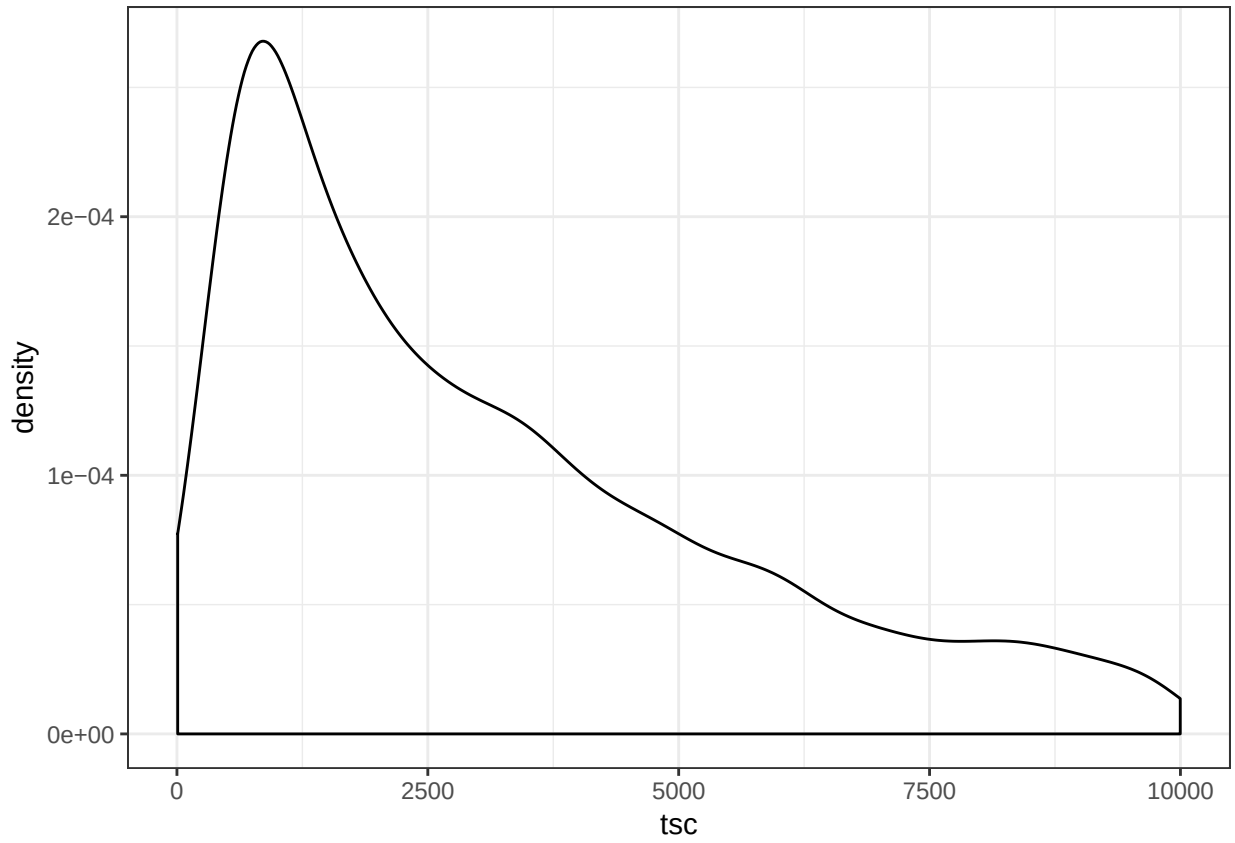
```
##                                                        title   author
## 1 Which logo do you like for a CBD/ Hemp Calming Chew Business? meloshii
##   edited is_self locked spoiler stickied score     id subreddit
## 1  False    True  False   False    False     1 brb738       Art
##                                                                                                  url
## 1 https://www.reddit.com/r/Art/comments/brb738/which_logo_do_you_like_for_a_cbd_hemp_calming/
##   num_comments   hot body    created   measured
## 1            0 False  404 1558449542 1558449591
```

Lets add some varaibles related to time such as: time since creation, hour of the day and day of the week.

```
dat$tsc<-dat$measured-dat$created
dat$hc<-substr(anytime(dat$created),12,13)
dat$wd<-weekdays(as.Date(substr(anytime(dat$created),1,20),'%Y-%m-%d'),abbreviate = TRUE)
```

In the following plot we can see that a usual post get to its hot section within more or less 1250 seconds since its creation.

```
ggplot(data = dat[dat$hot=="True",], aes_string(x = "tsc")) +      geom_density(alpha = 0.5)  +
    theme_bw()
```



Preprocessing the data

Dealing with the URLs

The objective of this function is to classify the file that its beeing posted. The alternatives are: images, videos, gifs, news, self and other.

```
urls<-as.data.frame(dat %>% group_by(id) %>% summarise(url=unique(url),isSelf=unique(as.integer(is_self
urls$isSelf<-ifelse(urls$isSelf==1,0,1)
whatIsIt<-function(dat){
  dat$isImage<-rep(0,nrow(dat))
  dat[grep("(.png)|(.jpg)",dat$url,ignore.case = TRUE),]$isImage<-1
  dat$isVideo<-rep(0,nrow(dat))
  dat[grep("(youtu.be)|(v.redd)|(.mp4)",dat$url,ignore.case = TRUE),]$isVideo<-1
  dat$isNews<-rep(0,nrow(dat))
  dat[grep("news",dat$url,ignore.case = TRUE),]$isNews<-1
  dat$isGif<-rep(0,nrow(dat))
  dat[grep("gif",dat$url,ignore.case = TRUE),]$isGif<-1
  dat$isOther<-rep(1,nrow(dat))
  dat$isOther<-ifelse((dat$isImage + dat$isVideo + dat$isNews + dat$isSelf + dat$isGif)>0,0,1)
  dat
}
```

Grouping rows of the same post

We are going to take the measurement score and number of comments closer to 5 and 10 minutes since the creation of the post. The following function does the former description. If your computer is able it is useful to compute it in parallel.

```
findCloser<-function(dat,time,paralelo=0){
  ids<-unique(dat$id)
  if(paralelo==1){
    library(foreach)
    library(doParallel)
    no_cores <- detectCores() - 2
    splits<-split(ids, ceiling(seq_along(ids)/no_cores))
    cl<-makeCluster(no_cores)
    registerDoParallel(cl)
    aux<-foreach(splite=splits,.combine = rbind) %dopar% {
    library(dplyr)
      ids<-splite
        res<-lapply(ids,function(x){
    aux<-dat[dat$id==x,]
    auxT<-lapply(time,function(y){
    row<-which.min(abs(aux$tsc-y))
    dist<-((aux[row,]$tsc-y))
    auxW<-aux[row,colnames(aux) %in% c("id","score","num_comments")]
    auxW<-auxW[,c(2,1,3)]
    auxW$dist<-dist
    colnames(auxW)<-c("id",paste("score",(y/60),sep="_"),paste("num_comments",(y/60),sep="_"),paste("dis
    auxW
    })
    aux<-bind_cols(auxT)
    aux
  })
  res<-bind_rows(res)
  res
    }
    res<-aux
  }
  else{
  res<-lapply(ids,function(x){
    aux<-dat[dat$id==x,]
    auxT<-lapply(time,function(y){
    row<-which.min(abs(aux$tsc-y))
    dist<-((aux[row,]$tsc-y))
    auxW<-aux[row,colnames(aux) %in% c("id","score","num_comments")]
    auxW<-auxW[,c(2,1,3)]
    auxW$dist<-dist
    colnames(auxW)<-c("id",paste("score",(y/60),sep="_"),paste("num_comments",(y/60),sep="_"),paste("dis
    auxW
    })
    aux<-bind_cols(auxT)
    aux
  })
  res<-bind_rows(res)
  }
```

```
  res
}
```

We will proceed to group our data by posts and we will take the following variables: the hour the post was created (hc), the day that was created (wd), the number of character of the title (title), the number of character of the body and the subreddit.

We will have 2 datasets, one for the classification task of predict wheter or not a post will get to hot; and another one to predict when will it get the hot section.

```
dat10<-findCloser(dat,c(300,600),paralelo=1)
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
dat10<-dat10[,-5]
datM<-as.data.frame(dat %>% group_by(id) %>% summarise(target=ifelse(any(hot=="True"),max(tsc),0),hc=uni

datF<-merge(dat10,datM,by.x="id",by.y="id")
datFC<-datF
datFC$hot<-ifelse(datFC$target>0,1,0)
```

Once we have grouped our data by posts we will get the information from the url described before aswell as som statistics from each subreddit.

```
urls<-whatIsIt(urls)
datF<-merge(datF,urls[,-2],by="id")

tip<-c("gif","self","image","video","news","other")

newDat<-as.data.frame(datF %>% group_by(subreddit) %>% summarise(min_score_5 = min(score_5),max_score_5

datF<-merge(datF,newDat,by="subreddit")

datF$isSelf<-ifelse(datF$isSelf==1,0,1)
datF$subreddit<-as.factor(datF$subreddit)
datF$hc<-as.integer(datF$hc)
datF$wd<-as.factor(datF$wd)
datF$isImage<-as.factor(as.character(datF$isImage))
datF$isSelf<-as.factor(as.character(datF$isSelf))
datF$isVideo<-as.factor(as.character(datF$isVideo))
datF$isNews<-as.factor(as.character(datF$isNews))
datF$isOther<-as.factor(as.character(datF$isOther))
datF$isGif<-as.factor(as.character(datF$isGif))
datF$tipFav<-as.factor(datF$tipFav)
```

We would need to save the classes of our variables aswell as the levels of the factors so that when we do live predictions we can easily transform the new data into the format of our trainning.

Processed training data

But first, lets take a look at the training data.

```
train<-fread("data/datFTr.csv",data.table = FALSE)
dim(train)
```

```
## [1] 91829    57
```

```
colnames(train)
```

```
##  [1] "id"                           "subreddit"
##  [3] "score_5"                      "num_comments_5"
##  [5] "dist_5"                       "score_10"
##  [7] "num_comments_10"              "dist_10"
##  [9] "target"                       "hc"
## [11] "wd"                           "title"
## [13] "body"                         "isSelf"
## [15] "isImage"                      "isVideo"
## [17] "isNews"                       "isGif"
## [19] "isOther"                      "min_score_5"
## [21] "max_score_5"                  "mean_score_5"
## [23] "sd_score_5"                   "quantile_score_5_0.25"
## [25] "quantile_score_5_0.5"         "quantile_score_5_0.75"
## [27] "min_num_comments_5"           "max_num_comments_5"
## [29] "mean_num_comments_5"          "sd_num_comments_5"
## [31] "quantile_num_comments_5_0.25" "quantile_num_comments_5_0.5"
## [33] "quantile_num_comments_5_0.75" "min_score_10"
## [35] "max_score_10"                 "mean_score_10"
## [37] "sd_score_10"                  "quantile_score_10_0.25"
## [39] "quantile_score_10_0.5"        "quantile_score_10_0.75"
## [41] "min_num_comments_10"          "max_num_comments_10"
## [43] "mean_num_comments_10"         "sd_num_comments_10"
## [45] "quantile_num_comments_10_0.25" "quantile_num_comments_10_0.5"
## [47] "quantile_num_comments_10_0.75" "tipFav"
## [49] "commentN"                     "commentK"
## [51] "commentR"                     "postN"
## [53] "postK"                        "postR"
## [55] "trophiesN"                    "has_verified_email"
## [57] "is_gold"
```

As we can see this dataset includes iformation from the user who posted the pst. We have decided not using this information due to the time it takes to scrap it.

As we have discussed before we have had to kept the classes and levels of our variables so that we always process our data in the same way.

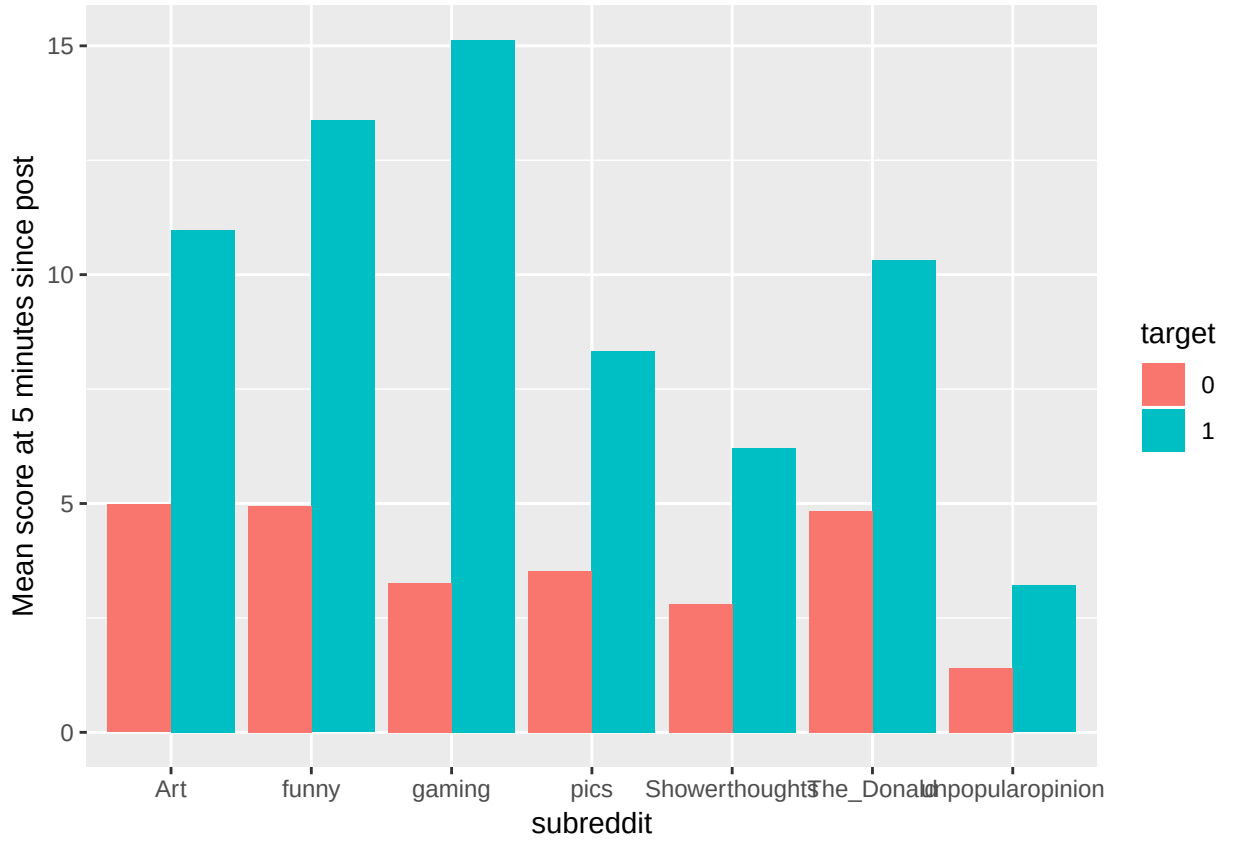
```
changeClass<-function(dat,vars,classes){
  for(i in 1:length(classes)){
    if(!(vars[i] %in% colnames(dat)))
        next
    if(classes[i]=="numeric"){
      dat[,vars[i]]<-as.numeric(as.character(dat[,vars[i]]))
```

```
    }
    if(classes[i]=="integer"){
      dat[,vars[i]]<-as.integer(as.character(dat[,vars[i]]))
    }
    if(classes[i]=="factor"){
      dat[,vars[i]]<-as.factor(as.character(dat[,vars[i]]))
    }
    if(classes[i]=="character"){
      dat[,vars[i]]<-as.character(dat[,vars[i]])
    }
  }
  dat
}

addLvlTrain<-function(dat){

  gifLvl<-readLines("data/isGiflevels.txt")
  imgLvl<-readLines("data/isImagelevels.txt")
  newLvl<-readLines("data/isNewslevels.txt")
  othLvl<-readLines("data/isOtherlevels.txt")
  slfLvl<-readLines("data/isSelflevels.txt")
  vidLvl<-readLines("data/isVideolevels.txt")
  subLvl<-readLines("data/subredditlevels.txt")
  tpfLvl<-readLines("data/tipFavlevels.txt")
  wkdLvl<-readLines("data/wdlevels.txt")
  levels(dat$isGif)<-gifLvl
  levels(dat$isImage)<-imgLvl
  levels(dat$isNews)<-newLvl
  levels(dat$isOther)<-othLvl
  levels(dat$isSelf)<-slfLvl
  levels(dat$isVideo)<-vidLvl
  levels(dat$subreddit)<-subLvl
  levels(dat$tipFav)<-tpfLvl
  levels(dat$wd)<-wkdLvl
  dat
}


vars<-readLines("data/dataVariables.txt")
classes<-readLines("data/dataClasses.txt")
train<-changeClass(train,vars,classes)
train<-addLvlTrain(train)
trainC<-train
trainC$target<-as.factor(ifelse(trainC$target>0,1,0))
```

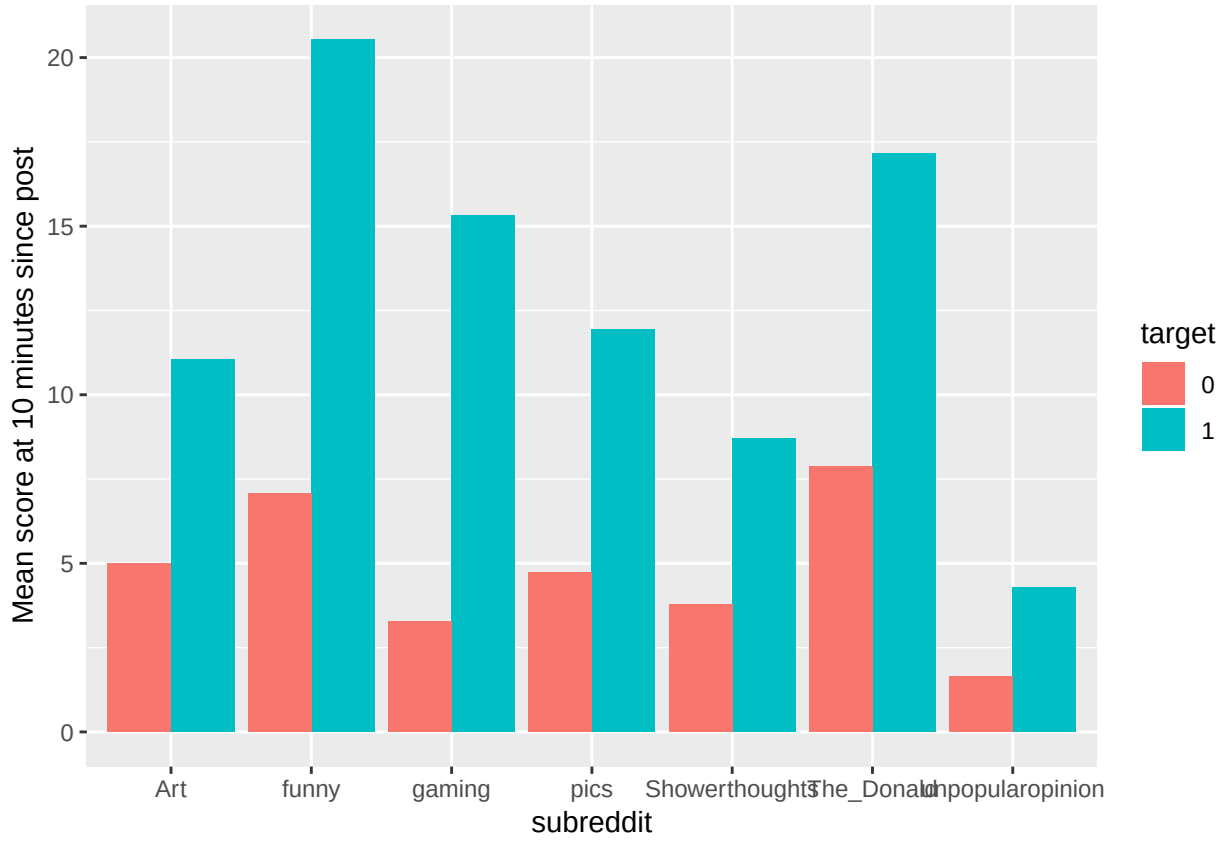
Plots of interest of processed data

Lets explore a little bit more our proccesed data

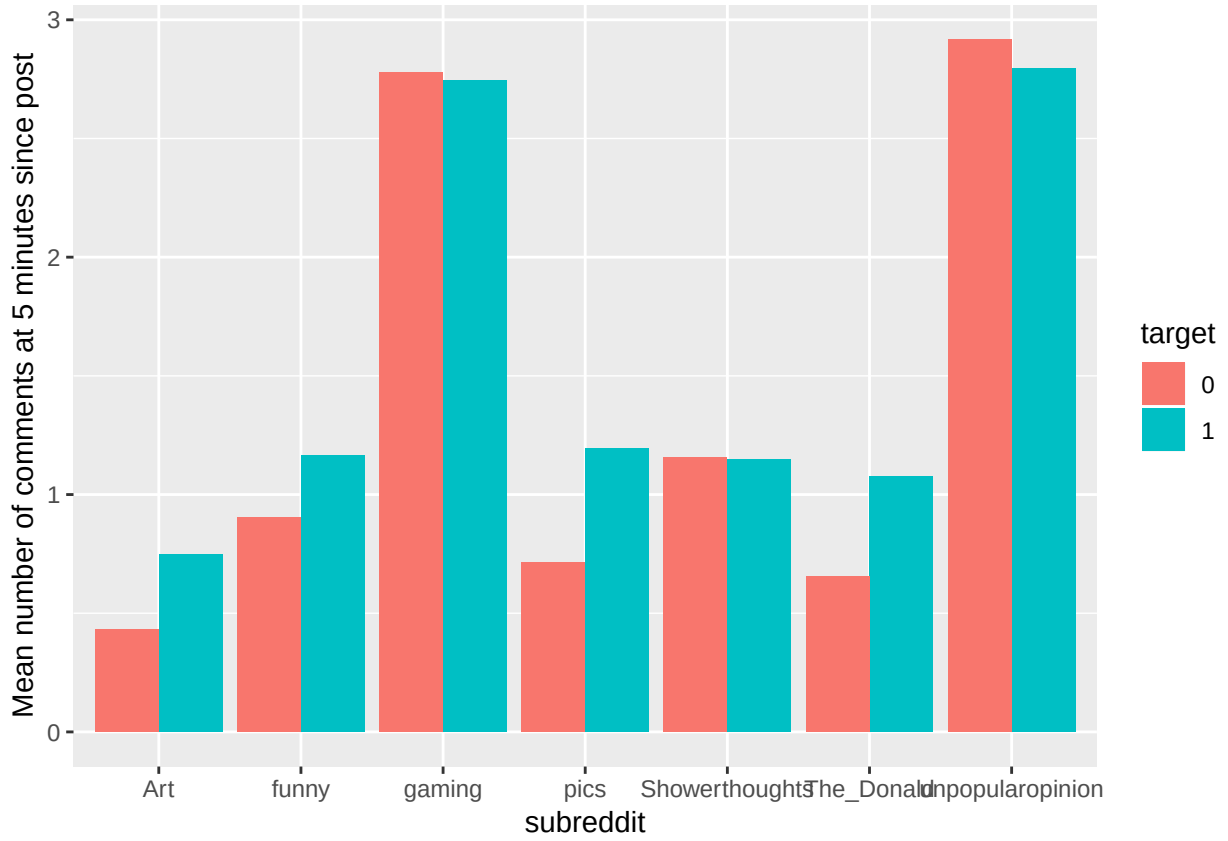
```
trainC[trainC$subreddit %in% unique(train$subreddit)[1:7],] %>% group_by(subreddit,target) %>% summaris
ggplot(aes(x = subreddit,y=value,fill=target))  +
    geom_bar(stat='identity', position='dodge') +
  labs(y="Mean score at 5 minutes since post")
```



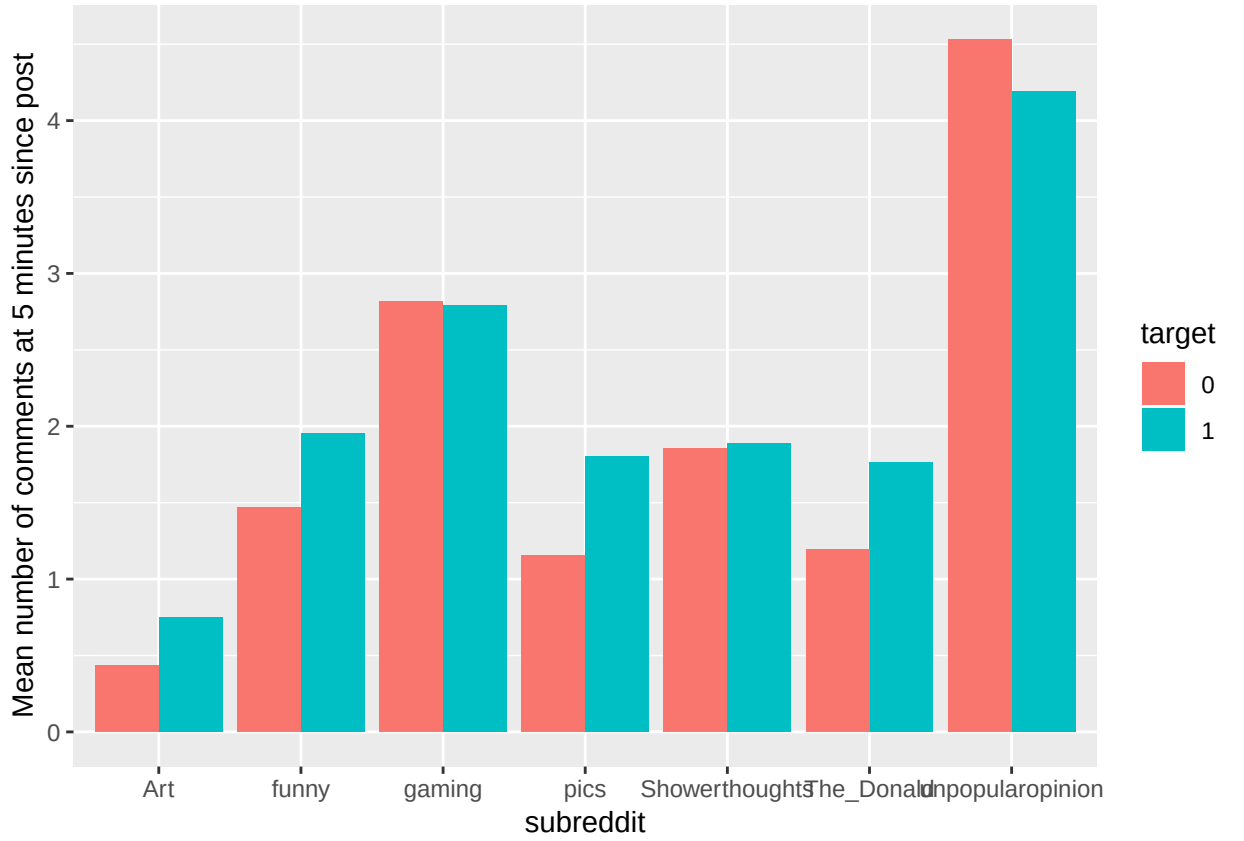

```
trainC[trainC$subreddit %in% unique(train$subreddit)[1:7],] %>% group_by(subreddit,target) %>% summaris
ggplot(aes(x = subreddit,y=value,fill=target))  +
    geom_bar(stat='identity', position='dodge') +
  labs(y="Mean score at 10 minutes since post")
```



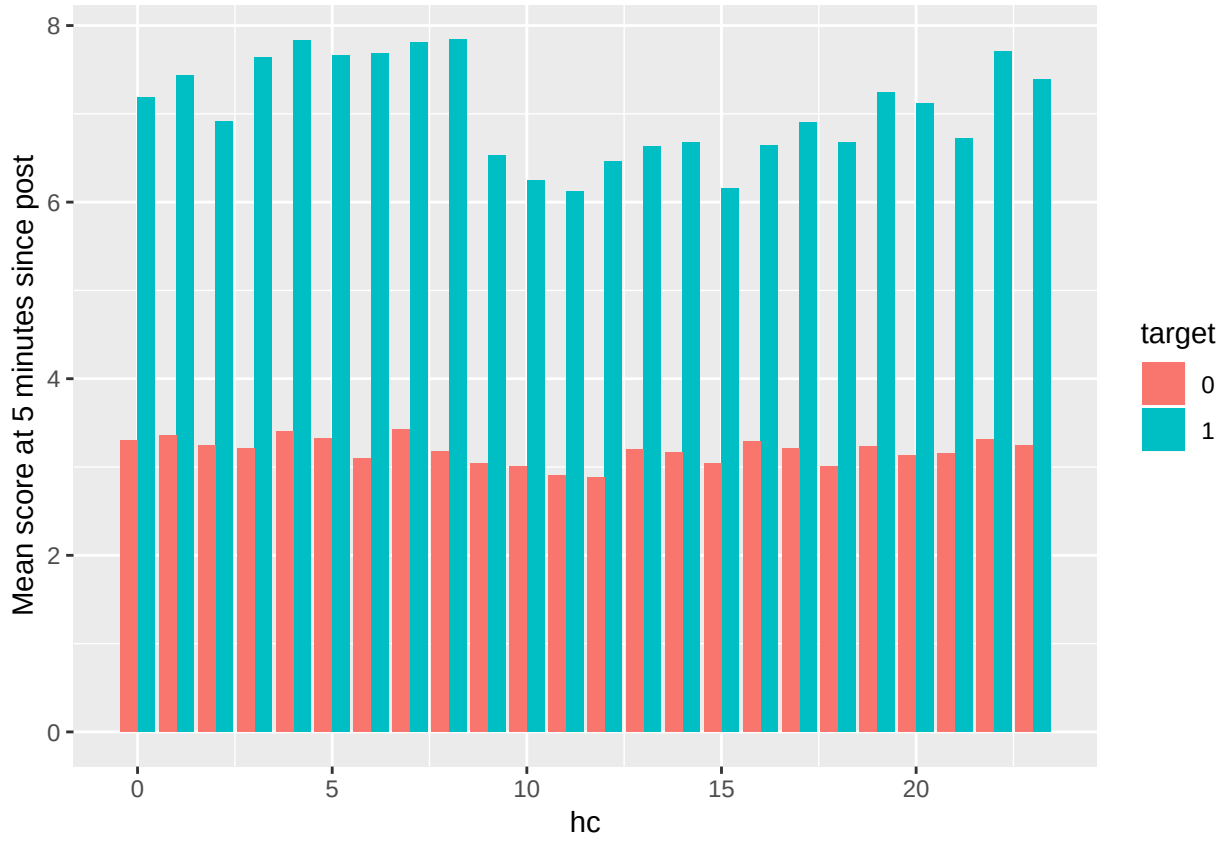

```
trainC[trainC$subreddit %in% unique(train$subreddit)[1:7],] %>% group_by(subreddit,target) %>% summaris
ggplot(aes(x = subreddit,y=value,fill=target))  +
    geom_bar(stat='identity', position='dodge') +
  labs(y="Mean number of comments at 5 minutes since post")
```



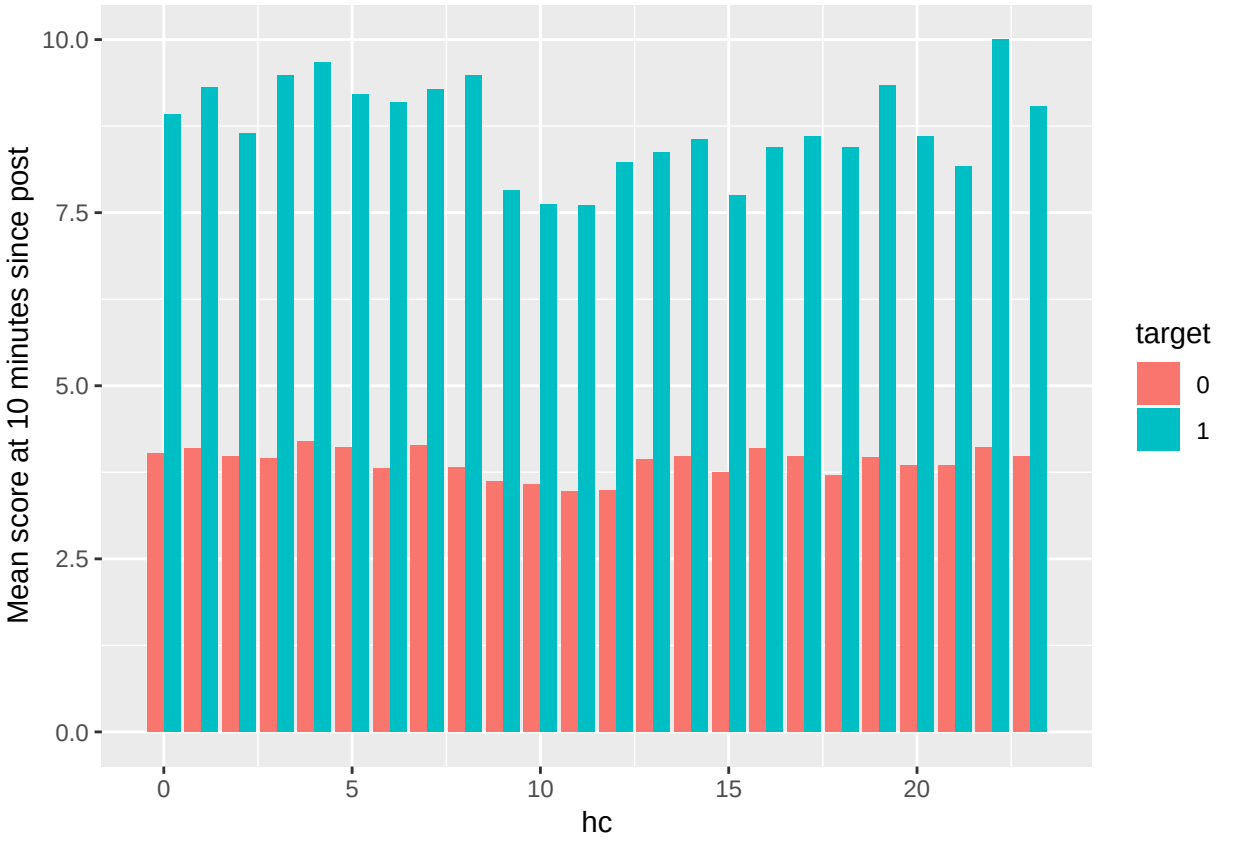

```
trainC[trainC$subreddit %in% unique(train$subreddit)[1:7],] %>% group_by(subreddit,target) %>% summaris
ggplot(aes(x = subreddit,y=value,fill=target))  +
    geom_bar(stat='identity', position='dodge') +
  labs(y="Mean number of comments at 5 minutes since post")
```




```
trainC %>% group_by(hc,target) %>% summarise(value=mean(score_5)) %>%
ggplot(aes(x = hc,y=value,fill=target))  +
    geom_bar(stat='identity', position='dodge') +
  labs(y="Mean score at 5 minutes since post")
```




```
trainC %>% group_by(hc,target) %>% summarise(value=mean(score_10)) %>%
ggplot(aes(x = hc,y=value,fill=target))  +
    geom_bar(stat='identity', position='dodge') +
  labs(y="Mean score at 10 minutes since post")
```



Creating the models

Having processed our data we are ready to search the parameters that best fit our data!

We will use an Extreme Gradient Boosting which hyperparameters will be searched using a genetic algorithm called Ant Colony Optimization.

The function below us are the "interface" that runs the cost functions for each model. We have created a custom metric which is based on the AUC but with an extra punishment to the false positive.

```
predXGBCMReg<-function(x,y,test,seed,paramList=c(0.3,1500,5,0,0.75,0.75),model=""){

  suppressMessages(library(xgboost))

  set.seed(seed)



  xgb_params <- list("objective" = "reg:squarederror",

                     "eval_metric" = "mae",

                     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 = rnorm(nrow(test)))



  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

}

predXGBCM<-function(x,y,test,seed,paramList=c(0.3,1500,5,0,0.75,0.75),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[

  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

}


costXGBAcoReg<-function(datos,paramList){
set.seed(123)
cv<-5
print(paramList)
print(Sys.time())
suppressMessages(library(dplyr))
  suppressMessages(library(MLmetrics))
  #aux<-cvDat(datos,id="id",cv=cv)
  aux<-datos
  aux$cvId<-sample(1:cv,nrow(datos),replace=TRUE)
  datosAux<-dumVar(datos[,!(colnames(datos) %in% c("target","id"))])
  fitness<-c()
  for(i in 1:cv){
    train<-datosAux[aux$cvId!=i,]
    test<-datosAux[aux$cvId==i,]
    train.y<-datos[aux$cvId!=i,]$target
    test.y<-datos[aux$cvId==i,]$target
    train<-train[,!(colnames(train) %in% c("target","id"))]
    test<-test[,!(colnames(test) %in% c("target","id"))]
    preds<-suppressMessages(predXGBCMReg(x = train,y = train.y,test = test,seed = 123,paramList=paramLis
```

```
    fitness<-c(fitness,MAE(preds,test.y))
  }
  print(mean(fitness))
  mean(fitness)
}

costXGBAco<-function(datos,paramList){
set.seed(123)
cv<-5
print(paramList)
print(Sys.time())
suppressMessages(library(dplyr))
  suppressMessages(library(MLmetrics))
  aux<-cvDat(datos,id="id",cv=cv)
  datosAux<-dumVar(datos[,!(colnames(datos) %in% c("target","id"))])

  fitness<-c()
  for(i in 1:cv){
    train<-datosAux[aux$cvId!=i,]
    test<-datosAux[aux$cvId==i,]
    train.y<-datos[aux$cvId!=i,]$target
    test.y<-datos[aux$cvId==i,]$target
    train<-train[,!(colnames(train) %in% c("target","id"))]
    test<-test[,!(colnames(test) %in% c("target","id"))]
    preds<-suppressMessages(predXGBCM(x = train,y = train.y,test = test,seed = 123,paramList=paramList))
    pFP<-sum(preds>0.5 & test.y==FALSE)/sum(preds>0.5)
    #P<-1-(sum(preds>0.5)/sum(test.y==TRUE))
    #metricM<-P+5*pFP
    metricM<-2*(1-AUC(preds,test.y))+pFP
    fitness<-c(fitness,metricM)
  }
  print(mean(fitness))
  mean(fitness)
}
```

```
trainC<-train
trainC$target<-(ifelse(trainC$target>0,1,0))
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(datF,costF=costXGBAco,hor=100,gen=24,tip=tip,paramListR=param)
#ACO(datFA,costF=costXGBAcoReg,hor=100,gen=24,tip=tip,paramListR=param)
costXGBAco(trainC,paramList = c(0.01,802,2,0,0.570719537472621,0.25))
```

```
## [1]   0.0100000 802.0000000   2.0000000   0.0000000   0.5707195   0.2500000
## [1] "2019-06-15 14:31:00 CEST"
## [1] 0.3308645

## [1] 0.3308645
```

```
trainR<-train[train$target>0,]

costXGBAcoReg(trainR,paramList = c(0.01,1185,7,0.09250,0.85952,0.588837))
```

```
## [1]    0.010000 1185.000000    7.000000    0.092500    0.859520    0.588837
## [1] "2019-06-15 14:31:19 CEST"
## [1] 1318.407
```

```
## [1] 1318.407
```

Once we have obtained the appropiated parameters we can save the models and have them ready for prediction.

We will use an already processed test data to check how does the model lo hace on outside trainning data.

```
test<-fread("data/datFTe.csv",data.table=FALSE)
test<-test[,-c(49:57)]
vars<-readLines("data/dataVariables.txt")
classes<-readLines("data/dataClasses.txt")

test<-changeClass(test,vars,classes)
test<-addLvlTrain(test)
new_ts <- model.matrix(~.+0,data = test[,-c(1,9)])
dtest <- xgb.DMatrix(data = new_ts,label = sample(c(TRUE,FALSE),nrow(test),replace=TRUE))

modelC<-xgb.load("data/modelC")
modelR<-xgb.load("data/modelR")

predsC<-predict(modelC,dtest)
predsR<-predict(modelR,dtest)

AUC(y_pred = predsC,y_true = ifelse(test$target>0,1,0))
```

```
## [1] 0.9081748
```

```
MAE(y_pred = predsR,y_true = test$target)
```

```
## [1] 4629.709
```

As we can see we have a relative good results on testing data. Despite this seemingly good results we have tested the bot on real time and we are having bad results due to time correlation.

Preprocessing test data on the go

We will use some python scripts to scrap the information from reddit. One of this scripts will execute an R script that will preprocess the data scrapped and make predictions. This R script can be found on /burningBot/writer/predictPost.r