# ML-Tools-Assignment-3

Gyongyver Kamenar (2103380)

4/12/2022

```
library(tidyverse)
library(caret)
library(MLmetrics)
library(xgboost)
library(pROC)
library(ggplot2)
library(kableExtra)
```

Get the data

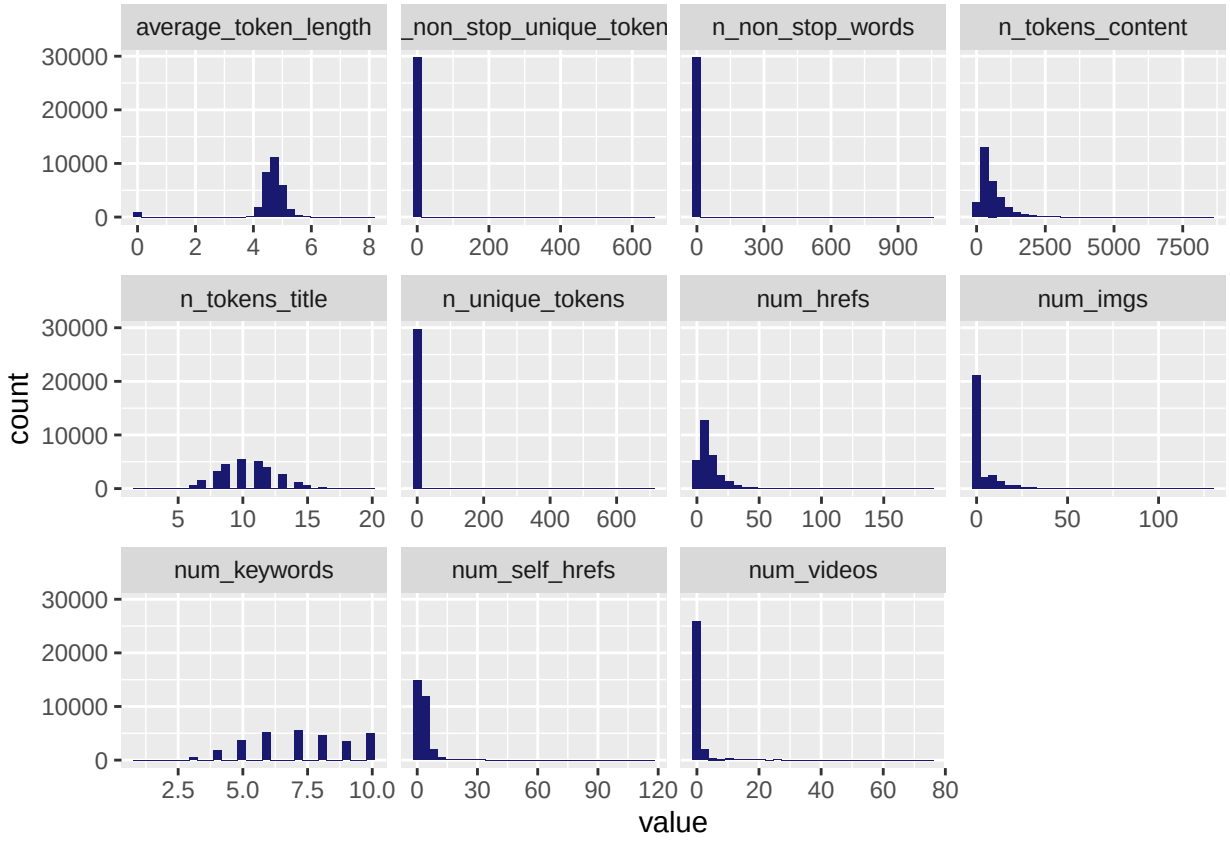
```
df<-read.csv("./online news data/train.csv")
test_submission<-read.csv("./online news data/test.csv")
sample_submission<-read.csv("./online news data/sample_submission.csv")
myseed<-20220412

set.seed(myseed)
```

## Exporatory data analysis

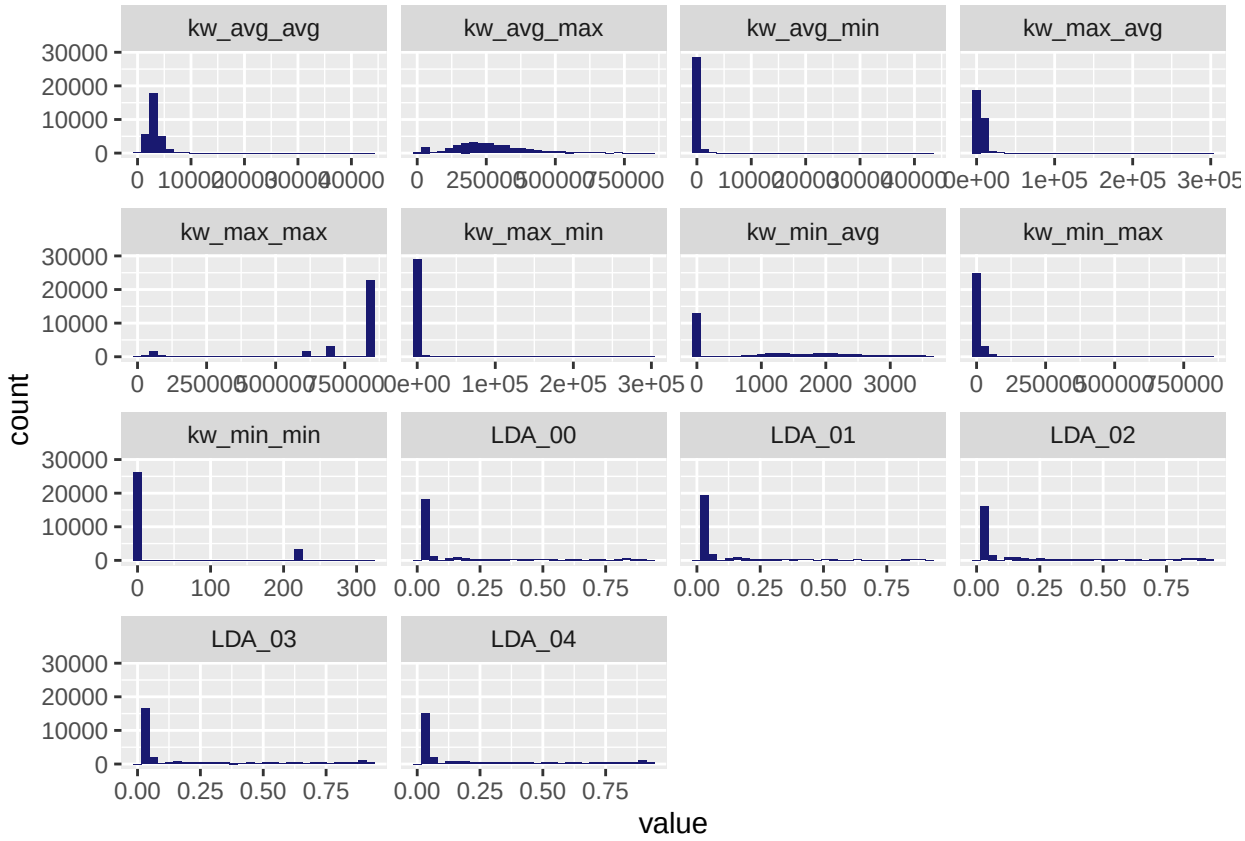
```
#str(df)

# Drop unnecessary columns: timedelta, article_id
df<-subset(df, select = -c(timedelta, article_id))

df<-df %>% mutate(
  is_popular = factor(is_popular, levels = list(1,0), labels = c('yes','no'))
)
```

### Visualization

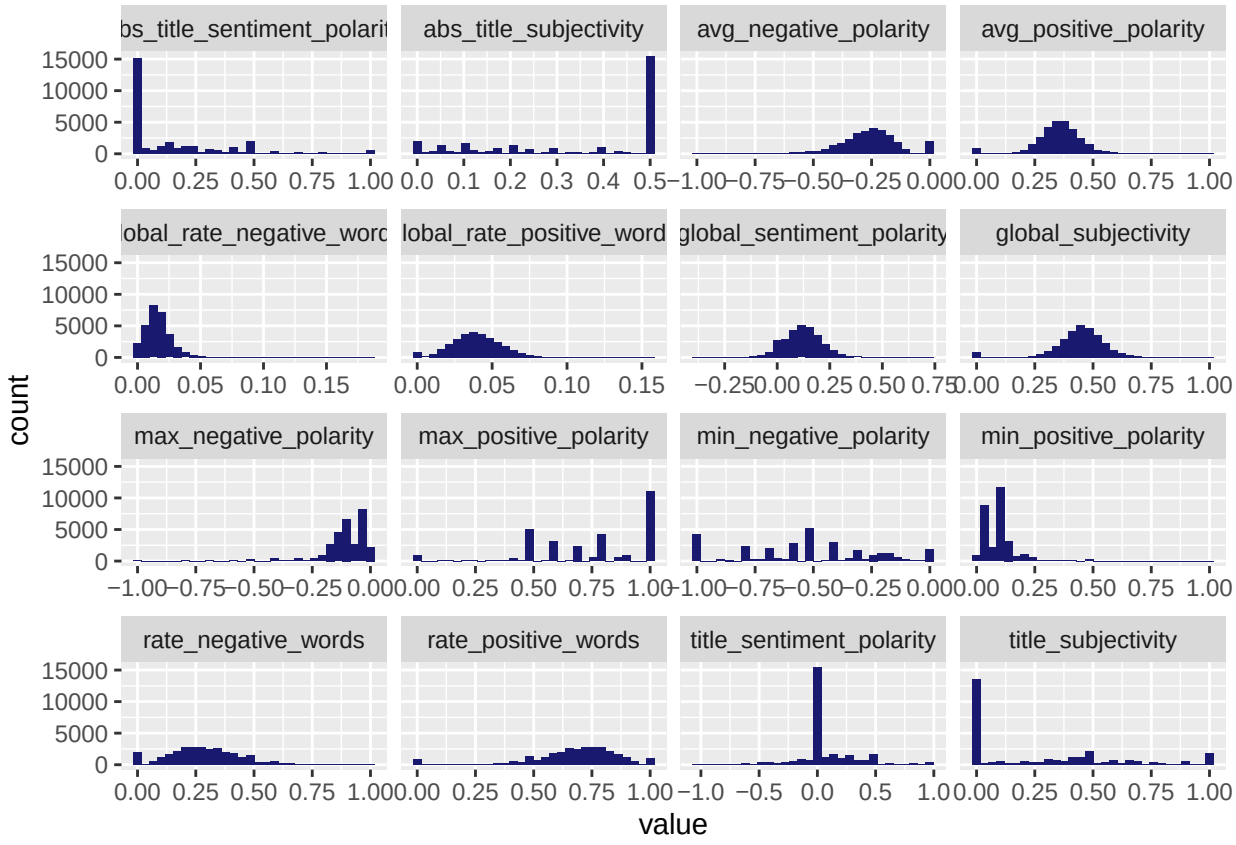
```
token_vars<-c("n_tokens_title" ,"n_tokens_content","n_unique_tokens",  "n_non_stop_words" , "n_non_stop_un

kw_vars<-c("kw_min_min" , "kw_max_min", "kw_avg_min"   , "kw_min_max" , "kw_max_max" , "kw_avg_max", "kw_m

sentiment_vars<-c("global_subjectivity" ,"global_sentiment_polarity", "global_rate_positive_words", "globa

plot_hist_facet<-function(df, vars){
  p<-subset(df, select = vars) %>% gather(vars) %>%  ggplot()+
    geom_histogram(aes(value), fill="midnightblue")+
    facet_wrap(~vars,scales = 'free_x')
  return(p)
}

plot_hist_facet(df,token_vars)
```




```
plot_hist_facet(df, kw_vars)
```



```
plot_hist_facet(df, sentiment_vars)
```




The dataset seems appropriate for modelling even at first glimpse. There are not missing values, disturbing extreme values, characters and so on. Even the original categorical variables like channel or weekday is converted into binary dummy variables, so it is appropriate for modelling. I train some basic model on this data, and if needed I will look for featrure engineering possibilities.

## Prepare modeling

Before modeling, I divide the data into train (85%) and test(15%) set.

```
# Train test split
ind <- sample(nrow(df),round(nrow(df)*0.85,0))
train<-df[ind,]
test<-df[-ind,]
```

## Logit model as benchmark

I trained a logit model on the train dataset with 5-fold cross-validation. I used 5-fold cross-validation in every model.

```
vars <- names(train[,1:(length(train)-1)])

form <- formula(paste0("is_popular ~", paste0(vars, collapse = " + ")))

ctrl <- trainControl(method = "cv",
                     number=5,
                     savePredictions = "final",
                     returnResamp = "final",
```

```
                    classProbs = TRUE,
                    summaryFunction = prSummary)

set.seed(myseed)

logit_model <- caret::train(
    form = form,
    method    = "glm",
    data      = train,
    family    = binomial,
    trControl = ctrl
  )

logit_model$results %>% kable() # AUC 0.231
```

| parameter | AUC | Precision | Recall | F | AUCSD | PrecisionSD | RecallSD | FSD |
|---|---|---|---|---|---|---|---|---|
| none | 0.2311328 | 0.3817974 | 0.0107944 | 0.0209724 | 0.0121638 | 0.1345161 | 0.0056594 | 0.0108611 |

```
Accuracy(logit_model$pred$pred,train$is_popular) %>% kable() # Accuracy 0.869
```

| x |
|---|
| 0.8693072 |

```
confusionMatrix(logit_model$pred$pred,train$is_popular) #too much false negative
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   yes    no
##        yes    14    75
##        no   3228 21956
##
##                Accuracy : 0.8693
##                  95% CI : (0.8651, 0.8734)
##     No Information Rate : 0.8717
##     P-Value [Acc > NIR] : 0.8762
##
##                   Kappa : 0.0016
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.004318
##             Specificity : 0.996596
##          Pos Pred Value : 0.157303
##          Neg Pred Value : 0.871823
##              Prevalence : 0.128279
##          Detection Rate : 0.000554
##    Detection Prevalence : 0.003522
##       Balanced Accuracy : 0.500457
##
##        'Positive' Class : yes
##
```

The accuracyo of the model seems good for first trial, but the AUC values is below 0.5 which meens that the model is worse then random. In the confusion matrix we can see, that the rate of false positive is really high.

## Lasso model

The next step is a Lasso regression with 50 different lambda values. I assume that with the penalty parameter, the Lasso will perform better that the logit.

```
lambda <- 10^seq(1, -4, length = 50)
grid <- expand.grid("alpha" = 1, lambda = lambda)

# Run LASSO
set.seed(myseed)
lasso_model <- train(form,
                     data = train,
                     method = "glmnet",
                     preProcess = c("center", "scale"),
                     trControl = ctrl,
                     tuneGrid = grid)
# Check the output
lasso_model$bestTune %>% kable()
```

| | alpha | lambda |
|---|---|---|
| 7 | 1 | 0.0004095 |

```
lasso_model$results %>% kable()
```

| alpha | lambda | AUC | Precision | Recall | F | AUCSD | PrecisionSD | RecallSD | FSD |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.0001000 | 0.2320921 | 0.3732326 | 0.0098685 | 0.0191960 | 0.0109530 | 0.1408772 | 0.0061227 | 0.0117482 |
| 1 | 0.0001265 | 0.2320815 | 0.3779427 | 0.0098685 | 0.0192075 | 0.0109236 | 0.1481239 | 0.0061227 | 0.0117720 |
| 1 | 0.0001600 | 0.2320650 | 0.3804427 | 0.0098685 | 0.0192103 | 0.0108621 | 0.1441585 | 0.0061227 | 0.0117690 |
| 1 | 0.0002024 | 0.2320397 | 0.3804427 | 0.0098685 | 0.0192103 | 0.0107997 | 0.1441585 | 0.0061227 | 0.0117690 |
| 1 | 0.0002560 | 0.2320622 | 0.3804427 | 0.0098685 | 0.0192103 | 0.0106934 | 0.1441585 | 0.0061227 | 0.0117690 |
| 1 | 0.0003237 | 0.2320676 | 0.3662338 | 0.0092517 | 0.0180242 | 0.0105673 | 0.1391100 | 0.0056598 | 0.0109003 |
| 1 | 0.0004095 | 0.2320927 | 0.3742180 | 0.0092517 | 0.0180236 | 0.0104454 | 0.1377320 | 0.0056598 | 0.0108784 |
| 1 | 0.0005179 | 0.2320502 | 0.3643478 | 0.0089430 | 0.0174221 | 0.0101936 | 0.1378550 | 0.0059063 | 0.0113577 |
| 1 | 0.0006551 | 0.2320357 | 0.3787068 | 0.0089430 | 0.0174313 | 0.0099857 | 0.1266468 | 0.0059063 | 0.0113505 |
| 1 | 0.0008286 | 0.2319849 | 0.3911422 | 0.0089430 | 0.0174483 | 0.0096448 | 0.1427765 | 0.0059063 | 0.0113745 |
| 1 | 0.0010481 | 0.2320048 | 0.3885781 | 0.0077094 | 0.0150802 | 0.0094585 | 0.1304154 | 0.0048714 | 0.0094146 |
| 1 | 0.0013257 | 0.2319169 | 0.4100233 | 0.0080180 | 0.0156890 | 0.0092182 | 0.1059978 | 0.0046723 | 0.0090194 |
| 1 | 0.0016768 | 0.2317027 | 0.3936045 | 0.0070931 | 0.0138998 | 0.0087031 | 0.1308677 | 0.0040186 | 0.0078027 |
| 1 | 0.0021210 | 0.2309754 | 0.4041711 | 0.0067854 | 0.0133201 | 0.0088411 | 0.1432808 | 0.0037131 | 0.0072353 |
| 1 | 0.0026827 | 0.2302229 | 0.4251010 | 0.0064772 | 0.0127378 | 0.0090550 | 0.2039541 | 0.0033436 | 0.0065527 |
| 1 | 0.0033932 | 0.2291675 | 0.4366667 | 0.0064772 | 0.0127433 | 0.0090381 | 0.1804316 | 0.0027582 | 0.0053994 |
| 1 | 0.0042919 | 0.2280333 | 0.4060606 | 0.0058604 | 0.0115439 | 0.0094432 | 0.1842235 | 0.0020105 | 0.0039596 |
| 1 | 0.0054287 | 0.2266723 | 0.4022145 | 0.0055522 | 0.0109388 | 0.0100435 | 0.2473151 | 0.0017596 | 0.0034888 |
| 1 | 0.0068665 | 0.2246596 | 0.4055844 | 0.0052436 | 0.0103356 | 0.0104324 | 0.2335360 | 0.0013795 | 0.0027292 |
| 1 | 0.0086851 | 0.2234830 | 0.3760317 | 0.0052436 | 0.0103354 | 0.0108567 | 0.1461852 | 0.0013795 | 0.0027288 |
| 1 | 0.0109854 | 0.2225429 | 0.3671429 | 0.0052436 | 0.0103317 | 0.0111127 | 0.1422846 | 0.0013795 | 0.0027257 |
| 1 | 0.0138950 | 0.2207540 | 0.3617460 | 0.0049349 | 0.0097276 | 0.0098979 | 0.1241454 | 0.0012893 | 0.0025376 |
| 1 | 0.0175751 | 0.2202241 | 0.4017460 | 0.0049349 | 0.0097378 | 0.0091408 | 0.1650378 | 0.0012893 | 0.0025457 |
| 1 | 0.0222300 | 0.2206500 | 0.3804396 | 0.0040090 | 0.0099053 | 0.0093302 | 0.2579311 | 0.0023377 | 0.0015276 |
| 1 | 0.0281177 | 0.2213495 | 0.4800000 | 0.0037004 | 0.0091709 | 0.0102193 | 0.3176301 | 0.0023372 | 0.0024857 |
| 1 | 0.0355648 | 0.2213693 | 0.2733333 | 0.0012336 | 0.0040916 | 0.0101772 | 0.3003701 | 0.0012909 | 0.0017778 |
| 1 | 0.0449843 | 0.2213693 | 0.3750000 | 0.0006168 | 0.0030769 | 0.0101772 | 0.4787136 | 0.0008446 | 0.0000067 |
| 1 | 0.0568987 | 0.0846849 | NaN | 0.0000000 | NaN | 0.1159612 | NA | 0.0000000 | NA |
| 1 | 0.0719686 | 0.0000000 | NaN | 0.0000000 | NaN | 0.0000000 | NA | 0.0000000 | NA |
| 1 | 0.0910298 | 0.0000000 | NaN | 0.0000000 | NaN | 0.0000000 | NA | 0.0000000 | NA |
| 1 | 0.1151395 | 0.0000000 | NaN | 0.0000000 | NaN | 0.0000000 | NA | 0.0000000 | NA |
| 1 | 0.1456348 | 0.0000000 | NaN | 0.0000000 | NaN | 0.0000000 | NA | 0.0000000 | NA |
| 1 | 0.1842070 | 0.0000000 | NaN | 0.0000000 | NaN | 0.0000000 | NA | 0.0000000 | NA |
| 1 | 0.2329952 | 0.0000000 | NaN | 0.0000000 | NaN | 0.0000000 | NA | 0.0000000 | NA |
| 1 | 0.2947052 | 0.0000000 | NaN | 0.0000000 | NaN | 0.0000000 | NA | 0.0000000 | NA |
| 1 | 0.3727594 | 0.0000000 | NaN | 0.0000000 | NaN | 0.0000000 | NA | 0.0000000 | NA |
| 1 | 0.4714866 | 0.0000000 | NaN | 0.0000000 | NaN | 0.0000000 | NA | 0.0000000 | NA |
| 1 | 0.5963623 | 0.0000000 | NaN | 0.0000000 | NaN | 0.0000000 | NA | 0.0000000 | NA |
| 1 | 0.7543120 | 0.0000000 | NaN | 0.0000000 | NaN | 0.0000000 | NA | 0.0000000 | NA |
| 1 | 0.9540955 | 0.0000000 | NaN | 0.0000000 | NaN | 0.0000000 | NA | 0.0000000 | NA |
| 1 | 1.2067926 | 0.0000000 | NaN | 0.0000000 | NaN | 0.0000000 | NA | 0.0000000 | NA |
| 1 | 1.5264180 | 0.0000000 | NaN | 0.0000000 | NaN | 0.0000000 | NA | 0.0000000 | NA |
| 1 | 1.9306977 | 0.0000000 | NaN | 0.0000000 | NaN | 0.0000000 | NA | 0.0000000 | NA |
| 1 | 2.4420531 | 0.0000000 | NaN | 0.0000000 | NaN | 0.0000000 | NA | 0.0000000 | NA |
| 1 | 3.0888436 | 0.0000000 | NaN | 0.0000000 | NaN | 0.0000000 | NA | 0.0000000 | NA |
| 1 | 3.9069399 | 0.0000000 | NaN | 0.0000000 | NaN | 0.0000000 | NA | 0.0000000 | NA |
| 1 | 4.9417134 | 0.0000000 | NaN | 0.0000000 | NaN | 0.0000000 | NA | 0.0000000 | NA |
| 1 | 6.2505519 | 0.0000000 | NaN | 0.0000000 | NaN | 0.0000000 | NA | 0.0000000 | NA |
| 1 | 7.9060432 | 0.0000000 | NaN | 0.0000000 | NaN | 0.0000000 | NA | 0.0000000 | NA |
| 1 | 10.0000000 | 0.0000000 | NaN | 0.0000000 | NaN | 0.0000000 | NA | 0.0000000 | NA |

```
Accuracy(lasso_model$pred$pred,train$is_popular) %>% kable()
```

| x |
|---|
| 0.8691489 |

The both the accuracy and the AUC values of the Lasso is similar to the logit model, so the penalty did not help much.

## Random forest

My next model is a random forest, with arbitrary hyperparameters.

```
# set tuning
tune_grid <- expand.grid(
  .mtry = 5, # c(5, 6, 7),
  .splitrule = "gini",
  .min.node.size = 10 # c(10, 15)
)
# By default ranger understands that the outcome is binary,
#   thus needs to use 'gini index' to decide split rule
set.seed(myseed)
rf_model_p <- caret::train(
  form,
  method = "ranger",
  data = train,
  tuneGrid = tune_grid,
  trControl = ctrl
)

# save model
save( rf_model_p , file = './Kaggle models/rf_model_1.RData' )

rf_model_p$results
```

```
# Load model
load("./Kaggle models/rf_model_1.RData")
rf_model_p$results %>% kable()
```

| mtry | splitrule | min.node.size | AUC | Precision | Recall | F | AUCSD | PrecisionSD | RecallSD | |
|---|---|---|---|---|---|---|---|---|---|---|
| 5 | gini | 10 | 0.2451539 | 0.3082051 | 0.0027754 | 0.0068532 | 0.0182399 | 0.1876391 | 0.0022857 | |

```
auc(as.numeric(test$is_popular),predict(rf_model_p,test, type = "prob")$yes)
```

```
## Area under the curve: 0.7369
```

The random forest's AUC is still ~0.23 which is unacceptable. I did not try further tuning on the random forest, because probably it would not help much.

## XGBoost

The next model I tried is XGBoost using the caret package. I used 5-fold cross validation and grid search on the dept of the tree and learning rate as you can see in the following code.

```
xgb_grid_1 = expand.grid(
  nrounds = 1000,
  max_depth = c(2, 4, 6, 8, 10), #depth of the tree 2
  eta=c(0.5, 0.1, 0.07), #learning rate 0.07
  gamma = 0.01, # minimum loss reduction
  colsample_bytree=0.5, # variables to choose from (%)
  min_child_weight=1,
  subsample=0.5
)
```

```
xgb_trcontrol_1 = trainControl(
  method = "cv",
  number = 5,
  verboseIter = FALSE,
  returnData = FALSE,
  returnResamp = "final", # save losses across all models
  classProbs = TRUE,  # set to TRUE for AUC to be computed
  summaryFunction = prSummary,
  allowParallel = TRUE
)

set.seed(myseed)
xgb_train_1 = train(
  x = as.matrix(train[ ,1:(length(train)-1)]),
  y = as.matrix(train$is_popular),
  trControl = xgb_trcontrol_1,
  tuneGrid = xgb_grid_1,
  method = "xgbTree"
)



xgb_train_1$results
xgb_train_1$bestTune



saveRDS(xgb_train_1,"./Kaggle models/xgb_train_1.rds")
#xgb_train_1<-readRDS("xgb_train_1.rds")



auc(as.numeric(test$is_popular),predict(xgb_train_1,test, type = "prob")$yes)
#predict(xgb_train_1,test, type = "prob")
Accuracy(predict(xgb_train_1,test),test$is_popular)

test_submission<-subset(test_submission, select = -c(timedelta, article_id))


sample_submission$score<-predict(xgb_train_1,test_submission,type="prob")$yes

write.csv(sample_submission,"sample_submission1.csv", row.names=FALSE)
```

```
xgb_train_1<-readRDS("./Kaggle models/xgb_train_1.rds")
auc(as.numeric(test$is_popular),predict(xgb_train_1,test, type = "prob")$yes) # test AUC 0.746
```

```
## Area under the curve: 0.746
```

```
Accuracy(predict(xgb_train_1,test),test$is_popular) # accuracy 0.868
```

```
## [1] 0.8681614
```

The xgboost reached 0.7460432 AUC on the test set. On the submission set, the model reached 0.713333 AUC.

## Train Xgb on full sample

I select the previous models best parameters based on the grid search and train a model on the full sample. This includes my test set too, so the test AUC value cannot be compared to the previous AUC but the additional data might reach better result on the submission test set.

```
xgb_grid_2 = expand.grid(
  nrounds = 1000,
  max_depth = 2, #depth of the tree 2
  eta=0.07, #learning rate 0.07
  gamma = 0.01, # minimum loss reduction
  colsample_bytree=0.5, # variables to choose from
  min_child_weight=1,
  subsample=0.5
)

xgb_trcontrol_2 = trainControl(
  method = "cv",
  number = 5,
  verboseIter = TRUE,
  returnData = FALSE,
  returnResamp = "final", # save losses across all models
  classProbs = TRUE,  # set to TRUE for AUC to be computed
  summaryFunction = prSummary,
  allowParallel = TRUE
)

set.seed(myseed)
xgb_train_full = caret::train(
  x = as.matrix(df[ ,1:(length(df)-1)]),
  y = as.matrix(df[,length(df)]),
  trControl = xgb_trcontrol_2,
  tuneGrid = xgb_grid_2,
  method = "xgbTree"
)


xgb_train_full$results # 0.9394
saveRDS(xgb_train_full,"./Kaggle models/xgb_train_full.rds")
sample_submission$score<-predict(xgb_train_1,test_submission,type="prob")$yes

write.csv(sample_submission,"sample_submission2.csv", row.names=FALSE)
```

```
xgb_train_full<-readRDS("Kaggle models/xgb_train_full.rds")
xgb_train_full$results # 0.9394
```

```
##   nrounds max_depth  eta gamma colsample_bytree min_child_weight subsample
## 1    1000         2 0.07  0.01              0.5                1       0.5
##         AUC Precision    Recall         F       AUCSD  PrecisionSD   RecallSD
## 1 0.9394006 0.8733781 0.9947513 0.9301212 0.002600979 0.0005927445 0.00183809
##             FSD
## 1 0.0009074811
```

```
auc(as.numeric(test$is_popular),predict(xgb_train_full,test, type = "prob")$yes)# 0.831
```

```
## Area under the curve: 0.831
```

```
Accuracy(predict(xgb_train_full,test ),test$is_popular)
```

```
## [1] 0.8748879
```

The model trained on the full data has obviously higher AUC on the training and 'fake test' sets, but actually the submission set's AUC did not change anything based on the public leaderboard.

## Tuning XGBoost

In the next step, I further tuned some parameters as you can see in the code below. - max dept: 2,4 - eta : 0.07,0.04,0.02 - gamma: 0.05, 0.07, 0.1 - colsample by tree: 0.3, 0.5, 0.7 - min child weight: 0.5, 1,2

```
xgb_grid_3 = expand.grid(
  nrounds = 1000,
  max_depth = c(2, 4), #depth of the tree 2
  eta=c(0.07, 0.04, 0.02), #learning rate 0.07
  gamma = c(0.05,0.07 ,0.1), # minimum loss reduction
  colsample_bytree=c(0.3,0.5,0.7), # variables to choose from (ratio)
  min_child_weight=c(0.5,1,2),
  subsample=0.5
)

xgb_trcontrol_1 = trainControl(
  method = "cv",
  number = 5,
  verboseIter = TRUE,
  returnData = FALSE,
  returnResamp = "final", # save losses across all models
  classProbs = TRUE,  # set to TRUE for AUC to be computed
  summaryFunction = prSummary,
  allowParallel = TRUE
)

set.seed(myseed)
xgb_train_2 = caret::train(
  x = as.matrix(train[ ,1:(length(train)-1)]),
  y = as.matrix(train$is_popular),
  trControl = xgb_trcontrol_1,
  tuneGrid = xgb_grid_3,
  method = "xgbTree"
)



xgb_train_2$results # AUC: 0.9382
xgb_train_2$bestTune

saveRDS(xgb_train_2,"./Kaggle models/xgb_train_2.rds")
#xgb_train_1<-readRDS("xgb_train_1.rds")
```

```
# Test
xgb_train_2<-readRDS("./Kaggle models/xgb_train_2.rds")
auc(as.numeric(test$is_popular),predict(xgb_train_2,test, type = "prob")$yes) # 0.7521
```

```
## Area under the curve: 0.7521
```

```
#predict(xgb_train_1,test, type = "prob")
Accuracy(predict(xgb_train_2,test),test$is_popular) # 0.8704
```

```
## [1] 0.8704036
```

```
# Predict submission sample
sample_submission$score<-predict(xgb_train_2,test_submission,type="prob")$yes

write.csv(sample_submission,"sample_submission3.csv", row.names=FALSE)
```

This model had higher AUC both on the train and test set than xgb_train_1. Also on the kaggle submission set, the model reached 0.7188 AUC.

Thereafter, I train a model on the full sample with the best parameter setting based on the previous grid search. This also increased the submission AUC to 0.71924.

```
xgb_grid_3 = expand.grid(
  nrounds = 1000,
  max_depth = c(4,6), #depth of the tree
  eta= 0.02, #learning rate
  gamma = 0.07, # minimum loss reduction
  colsample_bytree=0.7, # variables to choose from (ratio)
  min_child_weight=0.5,
  subsample=0.5
)



set.seed(myseed)
xgb_train_full2 = caret::train(
  x = as.matrix(df[ ,1:(length(df)-1)]),
  y = as.matrix(df[,length(df)]),
  trControl = xgb_trcontrol_1,
  tuneGrid = xgb_grid_3,
  method = "xgbTree"
)


saveRDS(xgb_train_full2,"./Kaggle models/xgb_train_full2.rds")

xgb_train_full2$results # AUC 0.940977

# Test
auc(as.numeric(test$is_popular),predict(xgb_train_full2,test, type = "prob")$yes) # AUC 0.8814

# Predict submission sample
sample_submission$score<-predict(xgb_train_full2,test_submission,type="prob")$yes
write.csv(sample_submission,"./Kaggle submission/sample_submission4.csv", row.names=FALSE)
```

```
xgb_train_full2<-readRDS("./Kaggle models/xgb_train_full2.rds")
xgb_train_full2$results # AUC 0.940977
```

```
##   nrounds max_depth  eta gamma colsample_bytree min_child_weight subsample
## 1    1000         4 0.02  0.07              0.7              0.5       0.5
##         AUC Precision    Recall         F       AUCSD  PrecisionSD     RecallSD
## 1 0.9409776  0.872607 0.9973758 0.9308289 0.003229036 0.0002653808 0.0006765366
##            FSD
## 1 0.0003338364
```

```
# Test
auc(as.numeric(test$is_popular),predict(xgb_train_full2,test, type = "prob")$yes)
```

```
## Area under the curve: 0.8814
```

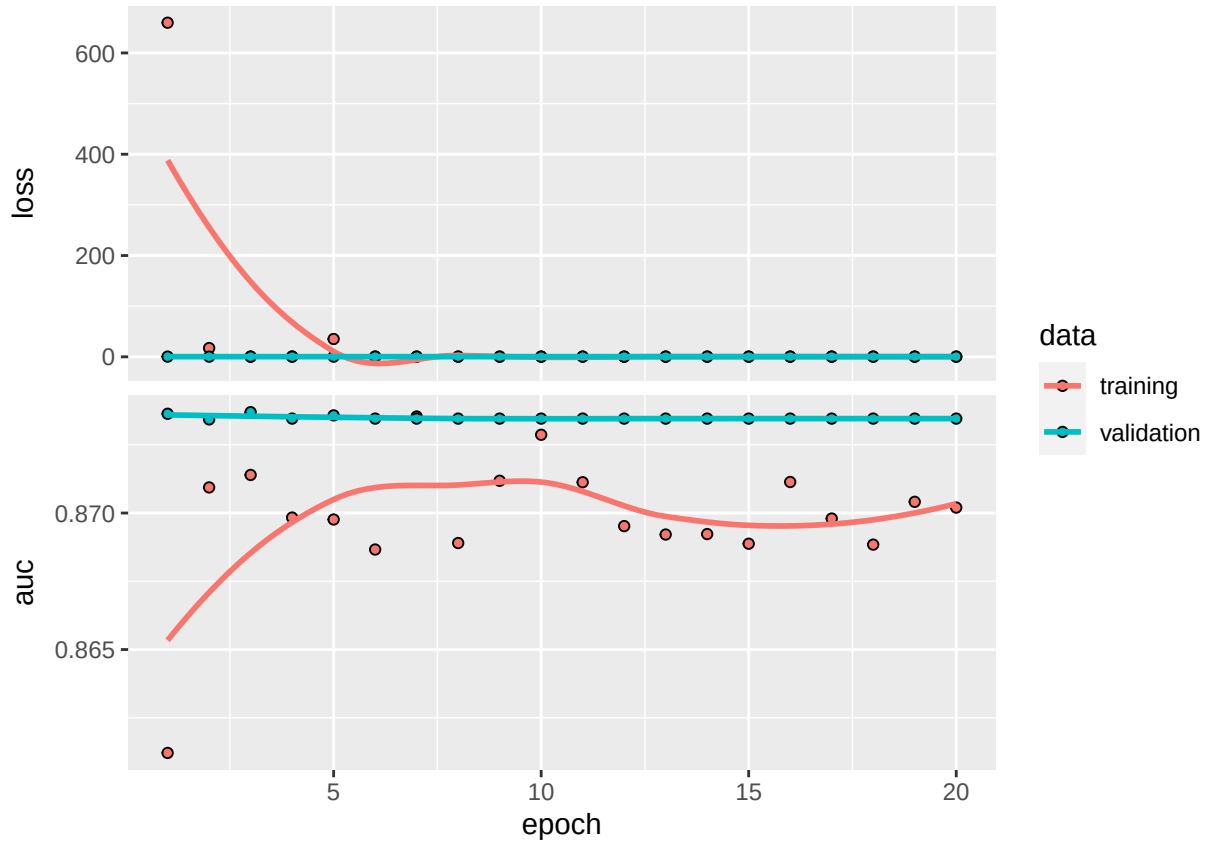
## Deeplearning with keras

To challenge the performance of deep learning, I train 2 neural networks with keras. The first one is a really simple model with 1 hidden layer.

```
library(keras)
library(tensorflow)
# Prepare data
set.seed(myseed)

keras_ind <- sample(nrow(train),round(nrow(train)*0.85,0))
keras_train<-train[keras_ind,]
keras_valid<-train[-keras_ind,]

train_x<-as.matrix(subset(keras_train,select = -is_popular))
popular<-c("yes","no")

train_y<-as.numeric(keras_train$is_popular)-1

valid_x<-as.matrix(subset(keras_valid,select = -is_popular))
valid_y<-as.numeric(keras_valid$is_popular)-1

test_x<-as.matrix(subset(test,select = -is_popular))
test_y<-as.numeric(test$is_popular)-1

test_submission_x<-as.matrix(subset(test_submission,select = -c(timedelta, article_id)))
# Reshape

train_y<-to_categorical(train_y,num_classes = 2)
valid_y<-to_categorical(valid_y,num_classes = 2)
test_y<-to_categorical(test_y,num_classes = 2)

# Build model

simple_keras <- keras_model_sequential()
simple_keras |>
    layer_dense(units = 128, activation = 'relu', input_shape = c(58)) |>
    layer_dropout(rate = 0.4) |>
    layer_dense(units = 2, activation = 'softmax')

summary(simple_keras)
```

```
## Model: "sequential"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  dense_1 (Dense)                    (None, 128)                     7552
##
##  dropout (Dropout)                  (None, 128)                     0
##
##  dense (Dense)                      (None, 2)                       258
##
## =========================================================================
## Total params: 7,810
## Trainable params: 7,810
## Non-trainable params: 0
## _________________________________________________________________________
```

```
library(tensorflow)
batch_size=8
compile(
    simple_keras,
    loss = 'binary_crossentropy',
    optimizer = optimizer_adam(lr=0.05), # learning rate
    metrics = tf$keras$metrics$AUC()
)

history1 <-fit(
    simple_keras,
    x=train_x,
    y=train_y,
    epochs = 20,
    batch_size = batch_size,
    #steps_per_epoch =floor(nrow(train_x)*0.85/batch_size),
    validation_split = 0.2,
    #validation_steps =floor(nrow(train_x)*0.15/batch_size)
)

simple_keras %>% save_model_hdf5("./Kaggle models/simple_keras.h5")

plot(history1)
```




```
#simple_keras<-load_model_hdf5("./Kaggle models/simple_keras.h5")
evaluate(simple_keras,valid_x,valid_y) # AUC 0.8752
```

```
##      loss       auc
## 0.3774098 0.8756260
```

```
evaluate(simple_keras,test_x,test_y) # AUC 0.8703
```

```
##      loss       auc
## 0.3868298 0.8703249
```

```
# prediction
auc(as.numeric(test_y),predict(simple_keras,test_x))
```

```
## Area under the curve: 0.8703
```

The model has ~0.87 accuracy on the validation and the test set. However, the values are not really changing. The prediction are the same for almost every observation.

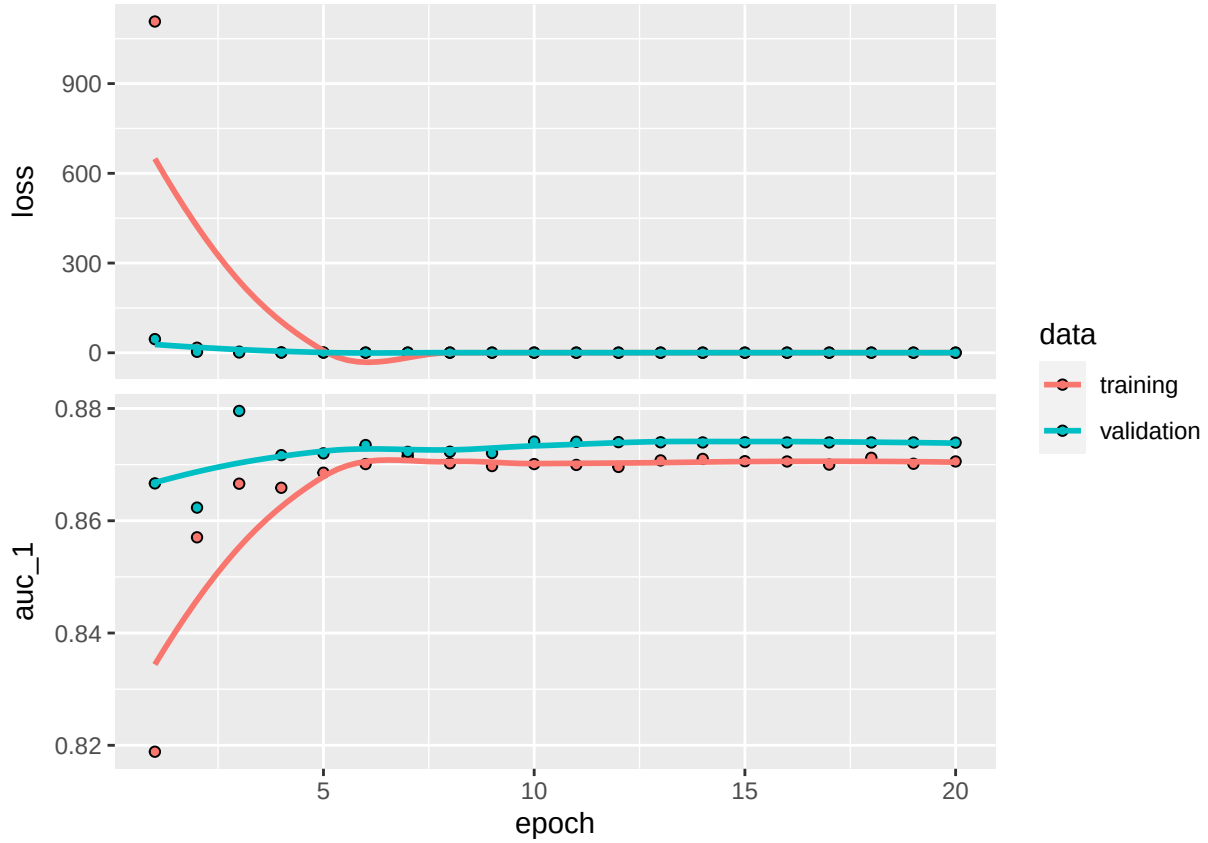
The second keras model has 2 hidden layers, but the problem with the prediction is the same. I tried several different tuning for the keras models, but it did not help. The models are not learning well.

```
keras_model_2 <- keras_model_sequential()
keras_model_2 |>
    layer_dense(units = 64, activation = 'relu', input_shape = c(58)) |>
    layer_dense(units = 128, activation = 'relu', input_shape = c(58)) |>
    layer_dropout(rate = 0.25) |>
    layer_dense(units = 2, activation = 'softmax')

summary(keras_model_2)
```

```
## Model: "sequential_1"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  dense_4 (Dense)                    (None, 64)                      3776
##
##  dense_3 (Dense)                    (None, 128)                     8320
##
##  dropout_1 (Dropout)                (None, 128)                     0
##
##  dense_2 (Dense)                    (None, 2)                       258
##
## =========================================================================
## Total params: 12,354
## Trainable params: 12,354
## Non-trainable params: 0
## _________________________________________________________________________
```

```
compile(
    keras_model_2,
    loss = 'binary_crossentropy',
    optimizer = optimizer_adam(),
    metrics = tf$keras$metrics$AUC()
)

set.seed(myseed)
batch_size=32

history2 <-fit(
    keras_model_2,
    train_x, train_y,
    epochs = 20,
    batch_size = batch_size,
    steps_per_epoch =floor(nrow(train_x)*0.8/batch_size),
```

```
    #validation_data = list(valid_x,valid_y),
    validation_split = 0.2,
    validation_steps =floor(nrow(train_x)*0.2/batch_size)
)

#keras_model_2 %>% save_model_hdf5("./Kaggle models/keras_model_2.h5")

plot(history2)
```




```
#keras_model_2<-load_model_hdf5("./Kaggle models/keras_model_2.h5", custom_objects = NULL, compile = TRUE)

auc(valid_y,predict(keras_model_2,valid_x)) # 0.8734
```

```
## Area under the curve: 0.8756
```

```
auc(test_y,predict(keras_model_2,test_x)) # 0.8686
```

```
## Area under the curve: 0.8706
```

## Stacking models

I wanted to test, whether the keras model improves the already tested xgboost with stacking. The public AUC on the kaggle submission test was just lower then just the xgboost.

```
sample_submission$score<-predict(keras_model_2,test_submission_x)[,2]*0.2+predict(xgb_train_full2,test_sub
#write.csv(sample_submission,"./Kaggle submission/sample_submission5.csv", row.names=FALSE)

sample_submission$score<-predict(xgb_train_full,test, type = "prob")$yes*0.2+predict(xgb_train_full2,test_
#write.csv(sample_submission,"./Kaggle submission/sample_submission6.csv", row.names=FALSE)
```

It did not improve the model.

## Improve XGBoost

To further improve the XGBoost model, I decreased the learning rate (eta) to 0.008 and increased the depth to 4 or 6. Then I trained this model on the full dataset.

```
xgb_grid_3 = expand.grid(
  nrounds = 1000,
  max_depth = c(4,6), #depth of the tree
  eta= 0.008, #learning rate
  gamma = 0.07, # minimum loss reduction
  colsample_bytree=0.7, # variables to choose from (ratio)
  min_child_weight=0.5,
  subsample=0.5
)


set.seed(myseed)
xgb_train_full3 = caret::train(
  x = as.matrix(df[ ,1:(length(df)-1)]),
  y = as.matrix(df[,length(df)]),
  trControl = xgb_trcontrol_1,
  tuneGrid = xgb_grid_3,
  method = "xgbTree"
)


saveRDS(xgb_train_full3,"./Kaggle models/xgb_train_full3.rds")

xgb_train_full3$results # AUC 0.9417

# Test
auc(as.numeric(test$is_popular),predict(xgb_train_full3,test, type = "prob")$yes) # AUC 0.916

# Predict submission sample
sample_submission$score<-predict(xgb_train_full3,test_submission,type="prob")$yes
write.csv(sample_submission,"./Kaggle submission/sample_submission7.csv", row.names=FALSE)

# On train sample

xgb_train_3 = caret::train(
  x = as.matrix(train[ ,1:(length(train)-1)]),
  y = as.matrix(train[,length(train)]),
  trControl = xgb_trcontrol_1,
  tuneGrid = xgb_grid_3,
  method = "xgbTree"
)


saveRDS(xgb_train_3,"./Kaggle models/xgb_train_3.rds")

xgb_train_3$results # AUC

# Test
auc(as.numeric(test$is_popular),predict(xgb_train_3,test, type = "prob")$yes)
```

```
# Predict submission sample
sample_submission$score<-predict(xgb_train_full3,test_submission,type="prob")$yes
write.csv(sample_submission,"./Kaggle submission/sample_submission8.csv", row.names=FALSE)
```

```
xgb_train_full3<-readRDS("./Kaggle models/xgb_train_full3.rds")
xgb_train_full3$results # AUC 0.9417
```

```
##     eta max_depth gamma colsample_bytree min_child_weight subsample nrounds
## 1 0.008         4  0.07              0.7              0.5       0.5    1000
## 2 0.008         6  0.07              0.7              0.5       0.5    1000
##         AUC Precision    Recall         F       AUCSD  PrecisionSD     RecallSD
## 1 0.9412976 0.8717597 0.9992667 0.9311684 0.002950672 0.0003262370 0.0004995268
## 2 0.9417075 0.8721556 0.9983405 0.9309917 0.002741461 0.0004476862 0.0003762247
##            FSD
## 1 0.0002085494
## 2 0.0002466909
```

```
xgb_train_full3$bestTune
```

```
##   nrounds max_depth   eta gamma colsample_bytree min_child_weight subsample
## 2    1000         6 0.008  0.07              0.7              0.5       0.5
```

```
# Test
auc(as.numeric(test$is_popular),predict(xgb_train_full3,test, type = "prob")$yes)
```

```
## Area under the curve: 0.916
```

```
# Variable importance
varImp(xgb_train_full3, scale=FALSE)
```

```
## xgbTree variable importance
##
##   only 20 most important variables shown (out of 58)
##
##                            Overall
## kw_avg_avg                 0.09871
## kw_max_avg                 0.05576
## average_token_length       0.03868
## self_reference_min_shares  0.03607
## self_reference_avg_sharess 0.03402
## kw_avg_max                 0.03374
## n_tokens_content           0.03075
## LDA_01                     0.02983
## global_subjectivity        0.02981
## LDA_03                     0.02971
## LDA_02                     0.02932
## kw_avg_min                 0.02920
## n_unique_tokens            0.02882
## LDA_00                     0.02876
## LDA_04                     0.02832
## n_non_stop_unique_tokens   0.02671
## kw_min_avg                 0.02601
## avg_positive_polarity      0.02503
## num_hrefs                  0.02472
## global_rate_positive_words 0.02451
```

```
xgb.importance(xgb_train_full3$finalModel$feature_names, model = xgb_train_full3$finalModel)
```

```
##                          Feature         Gain        Cover    Frequency
```

```
##  1:                   kw_avg_avg 9.870617e-02 1.068925e-01 4.460831e-02
##  2:                   kw_max_avg 5.576239e-02 6.262132e-02 3.798541e-02
##  3:          average_token_length 3.867868e-02 4.356679e-02 4.208621e-02
##  4:     self_reference_min_shares 3.606983e-02 5.081912e-02 2.909070e-02
##  5:    self_reference_avg_sharess 3.402078e-02 4.743026e-02 2.502840e-02
##  6:                   kw_avg_max 3.373669e-02 3.134573e-02 3.625267e-02
##  7:              n_tokens_content 3.075277e-02 2.967014e-02 3.338403e-02
##  8:                       LDA_01 2.982754e-02 2.479369e-02 3.313375e-02
##  9:           global_subjectivity 2.981219e-02 2.858849e-02 3.278720e-02
## 10:                       LDA_03 2.971047e-02 2.751730e-02 3.142027e-02
## 11:                       LDA_02 2.931616e-02 2.891444e-02 3.197859e-02
## 12:                   kw_avg_min 2.919577e-02 2.672884e-02 3.195934e-02
## 13:               n_unique_tokens 2.881573e-02 3.143105e-02 3.140101e-02
## 14:                       LDA_00 2.875732e-02 2.142134e-02 3.211336e-02
## 15:                       LDA_04 2.831998e-02 2.746105e-02 3.163204e-02
## 16:      n_non_stop_unique_tokens 2.670745e-02 2.500816e-02 2.957201e-02
## 17:                   kw_min_avg 2.601463e-02 3.280866e-02 2.495139e-02
## 18:         avg_positive_polarity 2.502665e-02 1.742083e-02 2.926397e-02
## 19:                     num_hrefs 2.471719e-02 2.734488e-02 2.629907e-02
## 20:    global_rate_positive_words 2.451405e-02 1.897707e-02 2.747348e-02
## 21:                   kw_max_min 2.264712e-02 1.773372e-02 2.601028e-02
## 22:     global_sentiment_polarity 2.143394e-02 1.481650e-02 2.495139e-02
## 23:                   kw_min_max 1.997239e-02 2.614524e-02 2.058104e-02
## 24:         avg_negative_polarity 1.952111e-02 1.241580e-02 2.331491e-02
## 25:    global_rate_negative_words 1.884297e-02 1.176212e-02 2.241004e-02
## 26:     self_reference_max_shares 1.834856e-02 1.589258e-02 2.100460e-02
## 27:           rate_positive_words 1.545288e-02 9.968531e-03 1.738511e-02
## 28:                      num_imgs 1.539364e-02 1.821444e-02 1.636472e-02
## 29:      title_sentiment_polarity 1.414482e-02 1.268812e-02 1.686529e-02
## 30:            title_subjectivity 1.276833e-02 1.014377e-02 1.443946e-02
## 31:        abs_title_subjectivity 1.137630e-02 8.284514e-03 1.270672e-02
## 32:                num_self_hrefs 1.118458e-02 1.188082e-02 1.353459e-02
## 33:                n_tokens_title 1.091577e-02 8.385747e-03 1.411217e-02
## 34:                    num_videos 1.091005e-02 1.818507e-02 1.133979e-02
## 35:  abs_title_sentiment_polarity 9.682582e-03 6.673867e-03 1.103175e-02
## 36:         min_positive_polarity 9.199646e-03 1.141452e-02 1.053118e-02
## 37:         max_negative_polarity 8.350352e-03 6.325475e-03 1.095474e-02
## 38:           rate_negative_words 7.810757e-03 8.565462e-03 8.759939e-03
## 39:         min_negative_polarity 7.515523e-03 3.530575e-03 9.144991e-03
## 40:         max_positive_polarity 5.458975e-03 2.839935e-03 6.314857e-03
## 41:                    is_weekend 5.372698e-03 1.013920e-02 4.986427e-03
## 42:                  num_keywords 4.439613e-03 2.750826e-03 5.544753e-03
## 43:                   kw_max_max 4.024442e-03 6.695634e-03 4.716890e-03
## 44:                   kw_min_min 3.878252e-03 4.706044e-03 4.312585e-03
## 45:        data_channel_is_socmed 2.761032e-03 3.472686e-03 2.714619e-03
## 46:             weekday_is_monday 2.616307e-03 4.134867e-03 3.061166e-03
## 47:           weekday_is_saturday 2.289214e-03 2.119104e-03 2.695366e-03
## 48: data_channel_is_entertainment 2.125969e-03 3.282688e-03 2.117787e-03
## 49:             weekday_is_sunday 2.120616e-03 3.743748e-03 2.483587e-03
## 50:          data_channel_is_tech 1.904253e-03 2.938080e-03 2.021524e-03
## 51:          weekday_is_wednesday 1.892067e-03 1.364612e-03 2.214051e-03
## 52:     data_channel_is_lifestyle 1.555481e-03 3.364233e-03 1.906009e-03
## 53:           data_channel_is_bus 1.309532e-03 1.667435e-03 1.424693e-03
## 54:            weekday_is_friday 1.192299e-03 1.068321e-03 1.597967e-03
## 55:           weekday_is_thursday 1.144088e-03 6.738613e-04 1.617220e-03
## 56:            weekday_is_tuesday 1.065126e-03 3.035109e-04 1.443946e-03
```

```
## 57:         data_channel_is_world 8.967556e-04 8.677835e-04 9.241255e-04
## 58:              n_non_stop_words 1.949014e-05 7.894469e-05 3.850523e-05
##                           Feature         Gain        Cover    Frequency
```

```
# train 3
xgb_train_3<-readRDS("./Kaggle models/xgb_train_full3.rds")
xgb_train_3$results
```

```
##     eta max_depth gamma colsample_bytree min_child_weight subsample nrounds
## 1 0.008         4  0.07              0.7              0.5       0.5    1000
## 2 0.008         6  0.07              0.7              0.5       0.5    1000
##         AUC Precision    Recall         F       AUCSD  PrecisionSD     RecallSD
## 1 0.9412976 0.8717597 0.9992667 0.9311684 0.002950672 0.0003262370 0.0004995268
## 2 0.9417075 0.8721556 0.9983405 0.9309917 0.002741461 0.0004476862 0.0003762247
##            FSD
## 1 0.0002085494
## 2 0.0002466909
```

This model performed well obviously on the train test, and also on the sumbission set. The model reached AUC 0.72241.

Then I tried increasing the depth of the tree:

```
xgb_grid_3 = expand.grid(
  nrounds = 1000,
  max_depth = c(8,10), #depth of the tree
  eta= 0.003, #learning rate
  gamma = 0.07, # minimum loss reduction
  colsample_bytree=0.7, # variables to choose from (ratio)
  min_child_weight=0.5,
  subsample=0.5
)
set.seed(myseed)
xgb_train_full4 = caret::train(
  x = as.matrix(df[ ,1:(length(df)-1)]),
  y = as.matrix(df[,length(df)]),
  trControl = xgb_trcontrol_1,
  tuneGrid = xgb_grid_3,
  method = "xgbTree"
)

saveRDS(xgb_train_full4,"./Kaggle models/xgb_train_full4.rds")
xgb_train_full4$results # AUC train0.9387 AUC full 0.9427
# Test
auc(as.numeric(test$is_popular),predict(xgb_train_full4,test, type = "prob")$yes) # AUC train 0.7477 AUC f
# Predict submission sample
sample_submission$score<-predict(xgb_train_full4,test_submission,type="prob")$yes
write.csv(sample_submission,"./Kaggle submission/sample_submission9.csv", row.names=FALSE)
```

Then I tried another grid search on just the train dataset, with several other max dept values , eta and colsamples as you can see below.

```
xgb_grid_3 = expand.grid(
  nrounds = 1000,
  max_depth = c(8,10,12,14), #depth of the tree c(6,7,8)
  eta= c(0.003,0.007,0.01), #learning rate
  gamma = 0.07, # minimum loss reduction
  colsample_bytree=c(0.7,0.5), # variables to choose from (ratio)
  min_child_weight=0.5,
```

```
  subsample=0.5
)



set.seed(myseed)
xgb_train_4 = caret::train(
  x = as.matrix(train[ ,1:(length(train)-1)]),
  y = as.matrix(train[,length(train)]),
  trControl = xgb_trcontrol_1,
  tuneGrid = xgb_grid_3,
  method = "xgbTree"
)



saveRDS(xgb_train_4,"./Kaggle models/xgb_train_4.rds")

xgb_train_4$bestTune # AUC train0.9396 AUC full
xgb_train_4$results
# Test
auc(as.numeric(test$is_popular),predict(xgb_train_4,test, type = "prob")$yes) # AUC 0.7546

# Predict submission sample
sample_submission$score<-predict(xgb_train_4,test_submission,type="prob")$yes
write.csv(sample_submission,"./Kaggle submission/sample_submission10.csv", row.names=FALSE)

varImp(xgb_train_full4, scale=FALSE)
xgb.importance(xgb_train_full4$finalModel$feature_names, model = xgb_train_full4$finalModel)
```

The train AUC is a bit lower then before, but is test is high compared to other models train on just the train set. The AUC on the test was 0.71875 which is slightly lower than it was before.

```
xgb_train_4<-readRDS("./Kaggle models/xgb_train_4.rds")
xgb_train_full4<-readRDS("./Kaggle models/xgb_train_full4.rds")
xgb_train_full4$results
```

```
##     eta max_depth gamma colsample_bytree min_child_weight subsample nrounds
## 1 0.003         8  0.07              0.7              0.5       0.5    1000
## 2 0.003        10  0.07              0.7              0.5       0.5    1000
##         AUC Precision    Recall         F       AUCSD  PrecisionSD     RecallSD
## 1 0.9402422 0.8717051 0.9993053 0.9311541 0.002524101 0.0000795133 0.0004001853
## 2 0.9402796 0.8718226 0.9993053 0.9312211 0.002819350 0.0002411278 0.0003761607
##            FSD
## 1 0.0001382675
## 2 0.0001458424
```

Then I trained the model on the full dataset as well with the best tune parameters of the previous model.

```
xgb_grid_3 = expand.grid(
  nrounds = 1000,
  max_depth = 8, #depth of the tree c(6,7,8)
  eta= 0.007, #learning rate
  gamma = 0.07, # minimum loss reduction
  colsample_bytree=0.7, # variables to choose from (ratio)
  min_child_weight=0.5,
  subsample=0.5
)
```

```
set.seed(myseed)
xgb_train_full5 = caret::train(
  x = as.matrix(df[ ,1:(length(df)-1)]),
  y = as.matrix(df[,length(df)]),
  trControl = xgb_trcontrol_1,
  tuneGrid = xgb_grid_3,
  method = "xgbTree"
)


saveRDS(xgb_train_full5,"./Kaggle models/xgb_train_full5.rds")


xgb_train_full5$results # AUC train0.9396 AUC full 0.94159
# Test
auc(as.numeric(test$is_popular),predict(xgb_train_full5,test, type = "prob")$yes) # AUC train 0.9781

# Predict submission sample
sample_submission$score<-predict(xgb_train_full5,test_submission,type="prob")$yes
write.csv(sample_submission,"./Kaggle submission/sample_submission11.csv", row.names=FALSE)

varImp(xgb_train_full5, scale=FALSE)
xgb.importance(xgb_train_full5$finalModel$feature_names, model = xgb_train_full5$finalModel)
```

It is clearly overfitted on the fake test set with 0.978 AUC, therefore it performed a bit worse on the submission set.

```
xgb_train_full5<-readRDS("./Kaggle models/xgb_train_full5.rds")
xgb_train_full5$results
```

```
##   nrounds max_depth   eta gamma colsample_bytree min_child_weight subsample
## 1    1000         8 0.007  0.07              0.7              0.5       0.5
##         AUC Precision    Recall         F       AUCSD  PrecisionSD     RecallSD
## 1 0.9415919 0.8720718 0.9983791 0.9309606 0.002563402 0.0003924624 0.0007424238
##           FSD
## 1 0.000220229
```

## Feature engineering

The data is very clean and structured as we can see on the histograms in the visualization section. However, some distibution as really skewed, similar to lognormal distribution, so I take the log of them to get closer to normal.

I also added some flags for variables, which has a given value with really high frequency compared to other values.

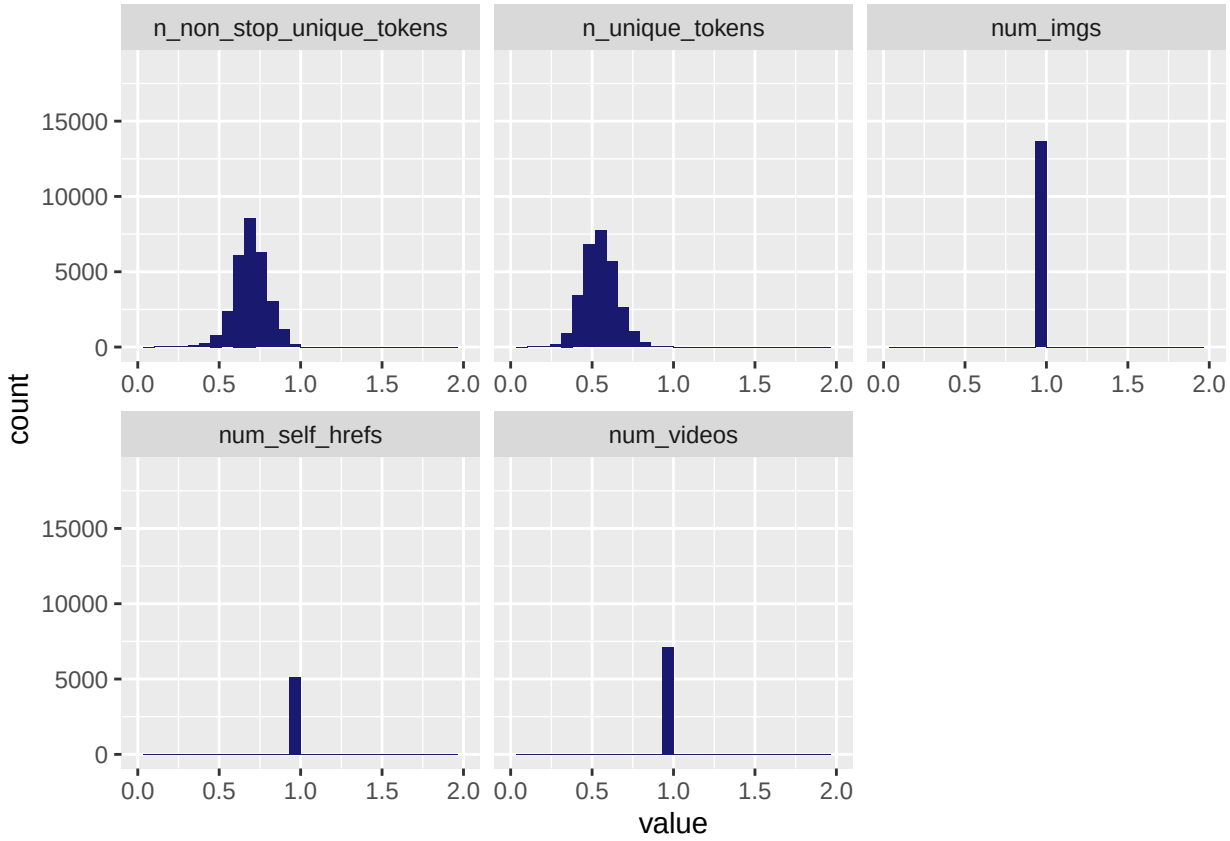
I used histograms of the distribution and the variable importance plots to select the variables.

```
library(reshape2)

names(df)
```

```
##  [1] "n_tokens_title"                "n_tokens_content"
##  [3] "n_unique_tokens"               "n_non_stop_words"
##  [5] "n_non_stop_unique_tokens"      "num_hrefs"
##  [7] "num_self_hrefs"                "num_imgs"
##  [9] "num_videos"                    "average_token_length"
## [11] "num_keywords"                  "data_channel_is_lifestyle"
## [13] "data_channel_is_entertainment" "data_channel_is_bus"
## [15] "data_channel_is_socmed"        "data_channel_is_tech"
## [17] "data_channel_is_world"         "kw_min_min"
```

```
## [19] "kw_max_min"                    "kw_avg_min"
## [21] "kw_min_max"                    "kw_max_max"
## [23] "kw_avg_max"                    "kw_min_avg"
## [25] "kw_max_avg"                    "kw_avg_avg"
## [27] "self_reference_min_shares"     "self_reference_max_shares"
## [29] "self_reference_avg_sharess"    "weekday_is_monday"
## [31] "weekday_is_tuesday"            "weekday_is_wednesday"
## [33] "weekday_is_thursday"           "weekday_is_friday"
## [35] "weekday_is_saturday"           "weekday_is_sunday"
## [37] "is_weekend"                    "LDA_00"
## [39] "LDA_01"                        "LDA_02"
## [41] "LDA_03"                        "LDA_04"
## [43] "global_subjectivity"           "global_sentiment_polarity"
## [45] "global_rate_positive_words"    "global_rate_negative_words"
## [47] "rate_positive_words"           "rate_negative_words"
## [49] "avg_positive_polarity"         "min_positive_polarity"
## [51] "max_positive_polarity"         "avg_negative_polarity"
## [53] "min_negative_polarity"         "max_negative_polarity"
## [55] "title_subjectivity"            "title_sentiment_polarity"
## [57] "abs_title_subjectivity"        "abs_title_sentiment_polarity"
## [59] "is_popular"
```

```
plot_hist_facet(df,c("n_non_stop_unique_tokens","n_unique_tokens","num_imgs","num_videos","num_self_hrefs"
  scale_x_continuous(limits = c(0,2))
```




```
# Calculating log values
df <-df %>% mutate(
  ln_n_non_stop_words = ifelse(n_non_stop_words==0,log(0.00000000000001),log(n_non_stop_words)),
  ln_n_unique_tokens = ifelse(n_unique_tokens==0,log(0.00000000000001),log(n_unique_tokens)),
  ln_num_videos = ifelse(num_videos==0,log(0.00000000000001),log(num_videos)),
```

```
  ln_num_self_hrefs = ifelse(num_self_hrefs==0,log(0.00000000000001),log(num_self_hrefs)),
  ln_kw_avg_min = ifelse(kw_avg_min==0,log(0.00000000000001),log(kw_avg_min)),
  ln_kw_max_avg = ifelse(kw_max_avg==0,log(0.00000000000001),log(kw_max_avg)),
  ln_kw_min_max = ifelse(kw_min_max==0,log(0.00000000000001),log(kw_min_max)),
  ln_LDA_00 = ifelse(LDA_00==0,log(0.00000000000001),log(LDA_00)),
  ln_LDA_01 = ifelse(LDA_01==0,log(0.00000000000001),log(LDA_01)),
  ln_LDA_02 = ifelse(LDA_02==0,log(0.00000000000001),log(LDA_02)),
  ln_LDA_03 = ifelse(LDA_03==0,log(0.00000000000001),log(LDA_03)),
  ln_LDA_04 = ifelse(LDA_04==0,log(0.00000000000001),log(LDA_04))


)

# Adding flags
df <- df %>% mutate(
  f_title_subjectivity =ifelse(title_subjectivity==0,1,0),
  f_title_sentiment_polarity =ifelse(title_sentiment_polarity==0,1,0),
  f_abs_title_subjectivity =ifelse(abs_title_subjectivity==0.5,1,0),
  f_abs_title_sentiment_polarity=ifelse(abs_title_sentiment_polarity==0,1,0),

)

# Apply the same on the submission  data
test_submission_fe <- test_submission %>% mutate(
  ln_n_non_stop_words = ifelse(n_non_stop_words==0,log(0.00000000000001),log(n_non_stop_words)),
  ln_n_unique_tokens = ifelse(n_unique_tokens==0,log(0.00000000000001),log(n_unique_tokens)),
  ln_num_videos = ifelse(num_videos==0,log(0.00000000000001),log(num_videos)),
  ln_num_self_hrefs = ifelse(num_self_hrefs==0,log(0.00000000000001),log(num_self_hrefs)),
  ln_kw_avg_min = ifelse(kw_avg_min==0,log(0.00000000000001),log(kw_avg_min)),
  ln_kw_max_avg = ifelse(kw_max_avg==0,log(0.00000000000001),log(kw_max_avg)),
  ln_kw_min_max = ifelse(kw_min_max==0,log(0.00000000000001),log(kw_min_max)),
  ln_LDA_00 = ifelse(LDA_00==0,log(0.00000000000001),log(LDA_00)),
  ln_LDA_01 = ifelse(LDA_01==0,log(0.00000000000001),log(LDA_01)),
  ln_LDA_02 = ifelse(LDA_02==0,log(0.00000000000001),log(LDA_02)),
  ln_LDA_03 = ifelse(LDA_03==0,log(0.00000000000001),log(LDA_03)),
  ln_LDA_04 = ifelse(LDA_04==0,log(0.00000000000001),log(LDA_04))
)

test_submission_fe <- test_submission_fe %>% mutate(
  f_title_subjectivity =ifelse(title_subjectivity==0,1,0),
  f_title_sentiment_polarity =ifelse(title_sentiment_polarity==0,1,0),
  f_abs_title_subjectivity =ifelse(abs_title_subjectivity==0.5,1,0),
  f_abs_title_sentiment_polarity=ifelse(abs_title_sentiment_polarity==0,1,0),

)
```

## Train XGBoost with feature engineered data

I trained XGBoost model on the feature engineered data.

```
train<-df[ind,]
test<-df[-ind,]

xgb_trcontrol_1 = trainControl(
  method = "cv",
  number = 5,
```

```
  verboseIter = TRUE,
  returnData = FALSE,
  returnResamp = "final", # save losses across all models
  classProbs = TRUE,  # set to TRUE for AUC to be computed
  summaryFunction = prSummary,
  allowParallel = TRUE
)

xgb_grid_3 = expand.grid(
  nrounds = 1000,
  max_depth = c(6,7), #depth of the tree
  eta= 0.008, #learning rate
  gamma = 0.07, # minimum loss reduction
  colsample_bytree=0.8, # variables to choose from (ratio)
  min_child_weight=0.5,
  subsample=0.5
)



set.seed(myseed)
xgb_train_5 = caret::train(
  x = as.matrix(subset(train, select = -is_popular)),
  y = as.matrix(train$is_popular),
  trControl = xgb_trcontrol_1,
  tuneGrid = xgb_grid_3,
  method = "xgbTree"
)


saveRDS(xgb_train_5,"./Kaggle models/xgb_train_5.rds")

xgb_train_5$results # AUC 0.9395

# Test
auc(as.numeric(test$is_popular),predict(xgb_train_5,test, type = "prob")$yes) # AUC 0.7581
# Predict submission sample
sample_submission$score<-predict(xgb_train_5,test_submission,type="prob")$yes
write.csv(sample_submission,"./Kaggle submission/sample_submission12.csv", row.names=FALSE)

# On full sample
set.seed(myseed)
xgb_train_full6 = caret::train(
  x = as.matrix(subset(df, select = -is_popular)),
  y = as.matrix(df$is_popular),
  trControl = xgb_trcontrol_1,
  tuneGrid = xgb_grid_3,
  method = "xgbTree"
)


saveRDS(xgb_train_full6,"./Kaggle models/xgb_train_full6.rds")

xgb_train_full6$results # AUC 0.9417
xgb_train_full6$bestTune
```

```
# Test
auc(as.numeric(test$is_popular),predict(xgb_train_full6,test, type = "prob")$yes) # AUC 0.919
# Predict submission sample
sample_submission$score<-predict(xgb_train_full6,test_submission,type="prob")$yes
write.csv(sample_submission,"./Kaggle submission/sample_submission13.csv", row.names=FALSE)
```

```
xgb_train_5<-readRDS("./Kaggle models/xgb_train_5.rds")
xgb_train_full6<-readRDS("./Kaggle models/xgb_train_full6.rds")
```

The models performed well, but not better than my best on the submission sample.

## Stacking all XGBoost models

Finally, I stacked all XGBoost models and it reached 0.72187 AUC on the kaggle leaderboard, which is my second best value. I did not outperform the xgb full 3 which has the best AUC out of my submissions.

```
sample_submission$score<-(predict(xgb_train_1,test_submission, type="prob")$yes+
                            predict(xgb_train_2,test_submission,type="prob")$yes+
                            predict(xgb_train_3,test_submission, type="prob")$yes+
                            predict(xgb_train_4,test_submission, type="prob")$yes+
                            predict(xgb_train_5,test_submission_fe, type="prob")$yes+
                            predict(xgb_train_full,test_submission, type="prob")$yes+
                            predict(xgb_train_full2,test_submission, type="prob")$yes+
  predict(xgb_train_full3,test_submission,type="prob")$yes+
  predict(xgb_train_full4,test_submission,type="prob")$yes+
  predict(xgb_train_full5,test_submission,type="prob")$yes+
  predict(xgb_train_full6,test_submission_fe,type="prob")$yes)/11


write.csv(sample_submission,"./Kaggle submission/sample_submission14.csv", row.names=FALSE)
```

## Conclusion

In this assignment I tried 5 different model types and tuning to predict, whether online news will be popular. I used cross-validation on all models, tried stacking and feature enginnering as well. The xgboost_train_full3 was my best model based on the Kaggle sumbission public AUC and I would use this model for real prediction.