Popularity Prediction

New York Times Blog Articles

Philippe Lambot

April 8, 2021

I. EXECUTIVE SUMMARY

In this project, a **94.3 % AUC** level has been reached in predicting news popularity on the validation set.

It has also provided **useful insights** about predictive impact from unstructured data, timing, classification and word count.

Natural Language Processing has been combined with **Machine Learning**. In

Machine Learning, *eXtreme Gradient Boosting* has proved somewhat more performing than *Random Forest*. Working on bootstrapped resample distributions has defeated overfitting, revealed true distributions and opened up the way to an ensemble solution that has slightly outperformed both individual models.

TAGS: popularity prediction, timing, news types, word count, headlines, snippets, AUC, ROC, Natural Language Processing, corpus, Document Term Matrix, bag of words, Machine Learning, binary classification, Random Forest, eXtreme Gradient Boosting, True Positive Rate (sensitivity), False Positive Rate, overfitting, bootstrapping iterations, resamples, density functions, distributions, ensemble solution

GITHUB: ***https://github.com/Dev-P-L/NLP-News-Popularity-Prediction***

II. FOREWORD about DATA: ACKNOWLEDGEMENT and LIMITATION of USE

This research project is based on Data provided by The New York Times for a Kaggle competition. Any use of it must comply with requirements from ***https://developer.nytimes.com***.

This project is lodged with the GitHub repository ***https://github.com/Dev-P-L/NLP-News-Popularity-Prediction*** and is comprised of four files:

- README.md,
- NLP_News-Popularity-Prediction.Rmd (all code),
- NLP_News-Popularity-Prediction.html (methodology, insights, results)
- and NLP_News-Popularity-Prediction.pdf (idem as previous but in PDF format).

III. DATA PREPARATION

A. Data Download

Let’s download data and have a look at the dataset. The downloading command is not comprised of *stringsAsFactors = FALSE* in order to get information about the levels of non-numeric features.

```
ds <- read.csv("ds.csv", encoding = "UTF-8") %>%
      select(- 1)
str(ds, vec.len = 1)
```

```
## 'data.frame':    6532 obs. of  10 variables:
##  $ NewsDesk      : chr  "Business" ...
##  $ SectionName   : chr  "Crosswords/Games" ...
##  $ SubsectionName: chr  "" ...
##  $ Headline      : chr  "More School Daze" ...
##  $ Snippet       : chr  "A puzzle from Ethan Cooper that reminds me that a bill
is due." ...
##  $ Abstract      : chr  "A puzzle from Ethan Cooper that reminds me that a bill
is due." ...
##  $ WordCount     : int  508 285 ...
##  $ PubDate       : chr  "2014-09-01 22:00:09" ...
##  $ Popular       : int  1 0 ...
##  $ UniqueID      : int  1 2 ...
```

This table deserves some debriefing.

B. Data Profiling

In the table above, there are 6,532 observations; this number suffices for Natural Language Processing and Machine Learning.

There is no missing value in the dataset.

```
# If there are missing values, a warning will be printed with the number of missing
values.
mv <- sum(is.na(ds))
check <- if(mv > 0)
{print(paste("Warning: there are ", mv, " missing values."))}

rm(mv, check)
```

The dataset is comprised of ten features:

- eight features are predictors,
- one feature, i.e. “Popular”, contains labels (“Popular” or “Unpopular”),
- the last feature is an identifier.

Three of the predictors are categorical variables: *NewsDesk*, *SectionName* and *SubsectionName*. They have a limited number of levels. But, in each of them, there is an unnamed level *""*. The name *Undefined* will be attributed to that level in each of these three categorical features.

```
# Downloading again the dataset without creating factors.
ds <- read.csv("ds.csv", encoding = "UTF-8", stringsAsFactors = FALSE) %>%
      select(- 1)

# Naming undefined level in the 3 categorical features.
ds$NewsDesk[ds$NewsDesk == ""] <- "Undefined"
ds$SectionName[ds$SectionName == ""] <- "Undefined"
ds$SubsectionName[ds$SubsectionName == ""] <- "Undefined"

# Converting the 3 categorical features into factors.
ds <- ds %>%
  mutate(NewsDesk = as.factor(NewsDesk)) %>%
  mutate(SectionName = as.factor(SectionName)) %>%
  mutate(SubsectionName = as.factor(SubsectionName))
```

There are three textual features: *Headline*, *Snippet* and *Abstract*.

Some snippets and abstracts are empty, as indicated by the level *""*. This will not be changed since the features *Snippet* and *Abstract* will have the status of character feature and not the status of factor.

In the three textual features, the number of levels is somewhat inferior to the whole number of observations. For the feature *Headline*, this indicates that a few headlines are identical but neither null nor missing. For the features *Snippet* and *Abstract*, this originates in the null values and possibly also in some redundancies.

From a content point of view, the snippet example is exactly the same as the abstract example. This needs examining in exploratory data analysis in section *V. EDA and NLP on TEXTUAL PREDICTORS from the TRAINING SET* hereunder.

Among features, there are also *WordCount*, *Popular* and *UniqueID*.

The feature *Popular* gives the labels, i.e. the dependent variable, the target. There are two classes: 1 (for blog articles rated as popular) and 0 (for blog articles rated as unpopular). The levels will be transposed into character levels: *Popular* and *Unpopular*.

```
DfNews <- ds %>%
  mutate(Popular = as.factor(gsub(1, "Popular", gsub(0, "Unpopular", Popular))))
```

C. Data Splitting

The dataset will be split into a validation set and a training set: the validation set will contain one third of the rows from the original dataset, the training set two thirds.

Even if some categories (from e.g. *SectionName*) are populated in the validation set but not in the training set, the corresponding rows remain in the validation set. This might lower AUC a little bit on the validation set but in real-life prediction new data could indeed contain new categories.

Now, let's perform exploratory data analysis, but only on the training set since no information from the validation set should contribute to modelling.

IV. EXPLORATORY DATA ANALYSIS (EDA) on LABELS from TRAINING SET

The positive class is "Popular". Let's have a look at the breakdown between positive and negative classes.

	Popular	Unpopular
Training Set	728	3 626

The positive class is clearly a minority, which justifies not relying on accuracy metric to measure predictive performance but on ROC AUC. Let's calculate prevalence of the positive class.

	Prevalence of Positive Class ("Popular")
Training Set	17 %

Prevalence of positive class is 17 %. Now, let's turn to unstructured data, i.e. to the three textual predictors.

V. EDA and NLP on TEXTUAL PREDICTORS from the TRAINING SET

A. EDA on Textual Predictors

1. Headlines

	ILLUSTRATIVE RANDOM SAMPLE OF HEADLINES
Training Set Row Number 1017	A Topsy-Turvy Take on Existence
Training Set Row Number 2177	Britain Begins Sale of Stake in Eurostar
Training Set Row Number 1533	Communication Failure at the Museum

These three examples of headlines show, as expected, some very tidy text. After this insight, we can dispense with some corrective measures in Natural Language Processing.

Let’s now turn to the corresponding random sample of abstracts.

2. Abstracts

	ILLUSTRATIVE RANDOM SAMPLE OF ABSTRACTS
Training Set Row Number 1017	**The American artist Joel Morrison sees a fragile order behind Mary Jo Bangs spectral meditation on an unseen photograph.**
Training Set Row Number 2177	**The plan to sell a 40 percent stake in the high-speed rail service is part of a broader push to lower the countrys public debt.**
Training Set Row Number 1533	**Metropolitan Diary: A museum visitor knew she was hard of hearing, but she didnt expect an exchange with a stranger to turn into mumbles and shrieks.**

Just as the random sample of headlines, this random sample of abstracts shows cleanness. The last piece of textual information consists of snippets.

3. Snippets

	ILLUSTRATIVE RANDOM SAMPLE OF SNIPPETS
Training Set Row Number 1017	**The American artist Joel Morrison sees a fragile order behind Mary Jo Bangs spectral meditation on an unseen photograph.**
Training Set Row Number 2177	**The plan to sell a 40 percent stake in the high-speed rail service is part of a broader push to lower the countrys public debt.**
Training Set Row Number 1533	**Metropolitan Diary: A museum visitor knew she was hard of hearing, but she didnt expect an exchange with a stranger to turn into mumbles and shrieks.**

The first and the third snippets are exactly the same as the corresponding abstracts, but, interestingly enough, the second one is a shortened version of the corresponding abstract, whose last words have been dropped. After some checking, it appears that some other abstracts have been shortened as well. How many snippets differ from abstracts because some words have been dropped at the end?

```
# Building up a response vector registering the snippets that differ.
v <- as.logical(rep("FALSE", length(trainNews$Snippet)))
for (i in 1:nrow(trainNews)) {
  v[i] <- identical(as.character(trainNews$Snippet[i]),
                    as.character(trainNews$Abstract[i]))
}

# Calculating the number of snippets that differ from corresponding abstracts.
nr_dif <- as.integer(sum(v == FALSE))

# Calculating the proportion of snippets that differ from abstracts.
prop_dif <- paste(round((nr_dif / length(v)) * 100, 1), "%", sep = " ")
```

Let’s visualize results.

	SNIPPETS DIFFERING FROM ABSTRACTS
Number in Training Set	**60**
Proportion in Training Set	**1.4 %**

Snippets differing from abstracts are a very small minority. The vast majority of snippets are strictly identical to abstracts. Consequently, only one out of the two predictors will be kept. Which one will it be? Preference will be given to snippets because they might impact more on readers.

B. Natural Language Processing

The following transformations will be performed on the training set:

- for each training set observation, headline and snippet will be amalgamated;
- a corpus will be built;
- letters will be lowercased;
- punctuation marks will be removed;
- stopwords (from the function stopwords()) will be removed;
- words will be stemmed;
- sentences will be tokenized into stemmed words in a Document Term Matrix;
- sparsity management will only keep tokens appearing in headlines or snippets from at least 2.5 % of blog articles.

```
# Combining headlines and snippets.
v <- 1:nrow(trainNews)
for (i in 1:nrow(trainNews)) {
  v[i] <- paste(trainNews$Headline[[i]],
                trainNews$Snippet[[i]], sep = " ")
}

# Corpus is created only on training reviews in order to avoid
# any interference between training reviews and validation reviews.
# Otherwise, tokens from validation set could (at least slightly) impact
# on token selection when applying the sparsity threshold.
corpus <- VCorpus(VectorSource(v))

# Lowercasing, removing punctuation and stopwords, stemming.
corpus <- tm_map(corpus, content_transformer(tolower))
corpus <- tm_map(corpus, removePunctuation)
corpus <- tm_map(corpus, removeWords, stopwords("english"))
corpus <- tm_map(corpus, stemDocument)

# Building up a bag of words in a Document Term Matrix.
dtm <- DocumentTermMatrix(corpus)

# Managing sparsity with sparsity threshold.
sparse <- removeSparseTerms(dtm, 0.975)

# Converting the Document Term Matrix into a matrix and then into a data frame.
sentSparse <- as.data.frame(as.matrix(sparse))

# Making all column names R-friendly.
colnames(sentSparse) <- make.names(colnames(sentSparse))

rm(v, i, corpus, dtm, sparse)
```

C. Word Cloud

Let’s have a look at a word cloud based on the Document Term Matrix to get some pre-attentive insights.

```
set.seed(1)
wordcloud(colnames(sentSparse), colSums(sentSparse), min.freq = 1,
          max.words = 50, random.order = FALSE, rot.per = 1/3,
          colors = brewer.pal(8, "Set1"), scale = c(4,.5))
```

In the word cloud above, we can see tokens such as *fashion*, *obama*, *presid*, *senat*, *billion*, *bank*, *million*, *work*, etc.

D. Inserting Textual Information into the Training Set

Let’s add the columns from the Document Term Matrix to the training set.

```
train <- cbind(trainNews, sentSparse) %>%
  select(- Headline, - Snippet, - Abstract, - UniqueID)

# Keeping data frame under specific name for further use.
sentSparse_train <- sentSparse
rm(trainNews, sentSparse)
```

VI. EDA on TIME, CLASSIFICATION and NUMERICAL PREDICTORS from TRAINING SET

A. Temporal Predictors

In the training set, the feature *PubDate* is a time series indicating date and time of publication of blog articles, including year, month, day, hour, minute and second. Let's decompose this time series and extract components.

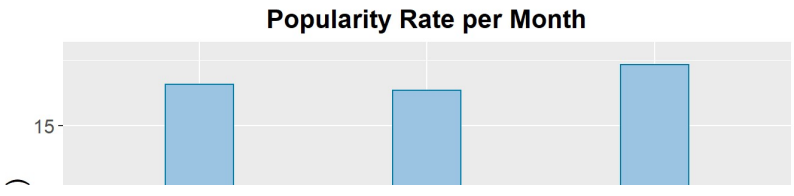
Year is not helpful here since there is no variation: all blog articles are from 2014.

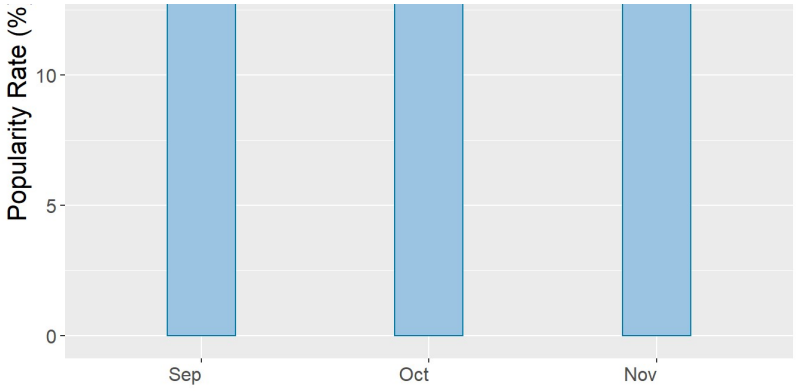
Month can be interesting: there are three of them, from September until November.

Two additional temporal patterns will be extracted: day of the week and hour in a 24-hour clock system.

```
temp <- train$PubDate %>%
  as.data.frame(stringsAsFactors = FALSE) %>%
  `colnames<-`("date") %>%
  mutate(date = as.POSIXct(date)) %>%
  mutate(weekday = weekdays(date, abbreviate = TRUE)) %>%
  mutate(month = month(date, label = TRUE)) %>%
  mutate(hour = hour(date)) %>%
  mutate(y = y_train)
```

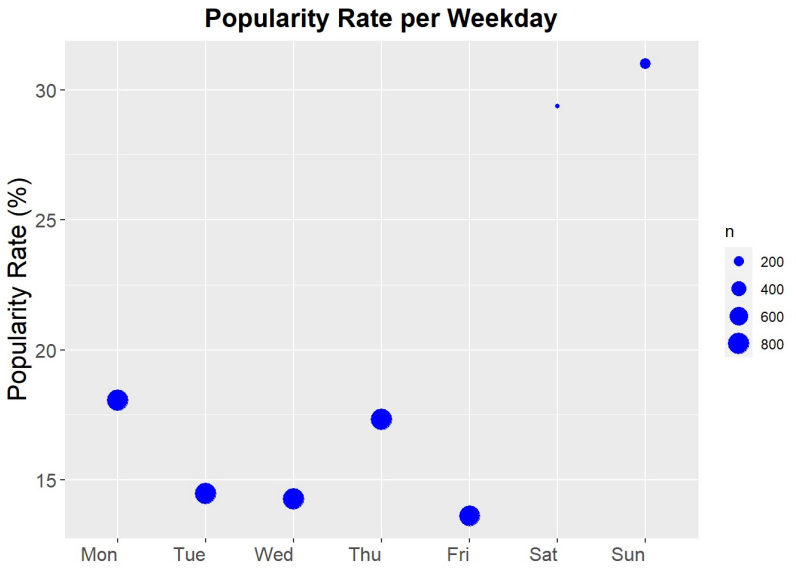
1. Average Popularity Rate per Month





Differences per month being Lilliputian, this predictor will not be added to the training set. Let's now have a try with weekdays.

2. Average Popularity Rate per Weekday



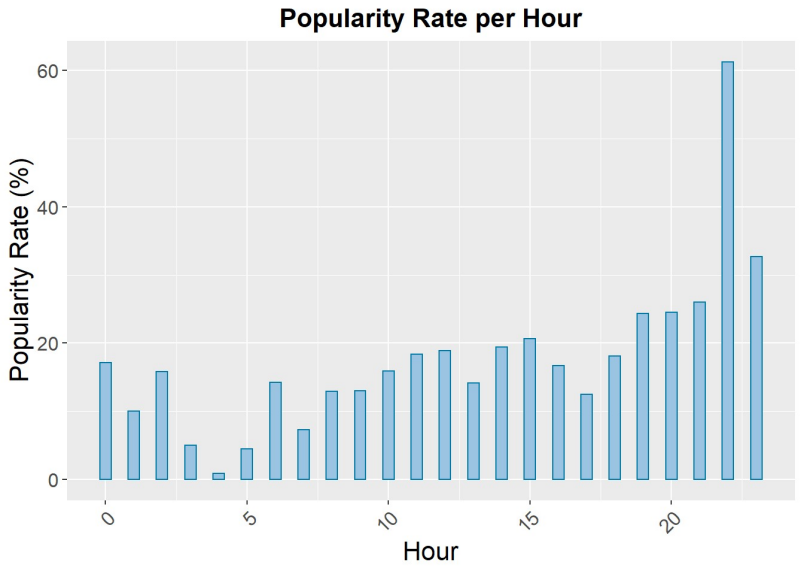
There is a clear difference between weekend and weekdays outside weekends: average popularity rate is substantially higher during weekend, and especially on Sundays. But are these higher rates statistically representative? It is representative for Sunday, with a number of blog articles around 200. But what about Saturday?

Weekday	Occurrence	Popularity Rate
Sun	216	31.0
Sat	126	29.4

Weekday	Occurrence	Popularity Rate
Mon	820	18.0
Thu	797	17.3
Tue	795	14.5
Wed	807	14.3
Fri	793	13.6

Saturday's higher popularity rate is also considered statistically significant with 126 blog articles.

3. Average Popularity Rate per Hour



There is an upward tendency in a 24-hour clock presentation: popularity rate is rather low between 3 a.m. and 5 a.m., then rises, especially from 8 p.m. onwards, culminating at more than 60 % in the 10 p.m. period.

But are the lowest and highest average popularity rates statistically representative? Let's have a look at a breakdown of observations per hour in a 24-hour clock presentation.

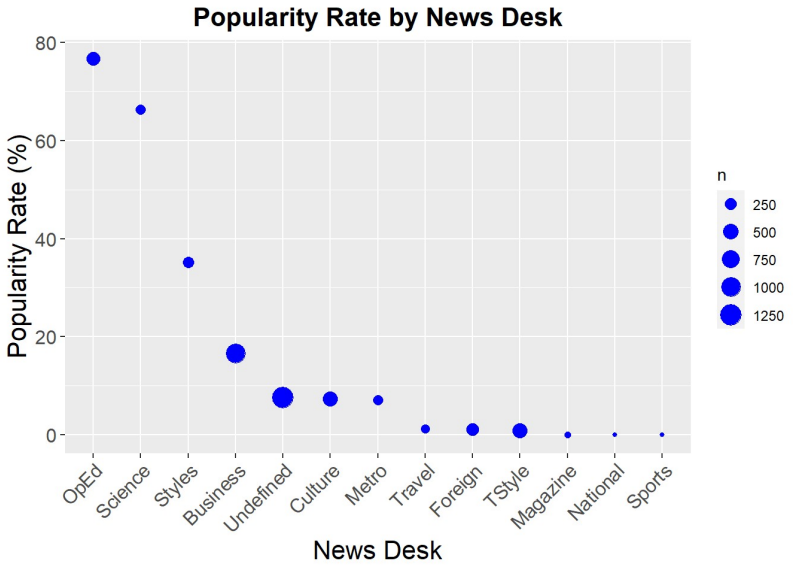
Hour	Occurrence	Popularity Rate
22	93	61.3 %
23	49	32.7 %
21	73	26 %
20	106	24.5 %
19	119	24.4 %
15	295	20.7 %
14	299	19.4 %
12	350	18.9 %
11	332	18.4 %
18	232	18.1 %
0	76	17.1 %
16	306	16.7 %
10	245	15.9 %
2	19	15.8 %
6	155	14.2 %
13	304	14.1 %
9	216	13 %
8	232	12.9 %
17	264	12.5 %
1	20	10 %
7	260	7.3 %
3	40	5 %
5	158	4.4 %
4	111	0.9 %

Yes, indeed, extreme rates are statistically representative. This is the case for the highest rate of more than 60 % with 93 instances. This is also the case for the lowest rates below 6 % if they are considered together. Consequently, this predictor will also be included into the training set.

Let's go on with some data profiling. Two predictors will be added to the training set: *weekday* (days of the week) and *hour* (hour in a 24-hour clock).

B. Classification Predictors

1. *News Desks*



The categories with average popularity rate below 20 % predominate: there are more *news desk* groups below 20 % than above and they are altogether much more populated, with the *undefined* group containing around 1,250 blog articles and the *business* group around 1,000.

Nevertheless, three *news desks* stand out from the majority and have much higher average popularity rates; among them the *op-ed* group has an average popularity rate of almost 80 %. Moreover, the *op-ed* average popularity rate can be considered statistically representative. Let's digitize information in the following table in order to get more precise information, among others about the numbers of blog articles in the *science* and *styles* categories.

News Desk	Occurrence	Popularity Rate

News Desk	Occurrence	Popularity Rate
OpEd	348	76.7 %
Science	125	66.4 %
Styles	196	35.2 %
Business	988	16.6 %
Undefined	1 250	7.5 %
Culture	447	7.4 %
Metro	140	7.1 %
Travel	79	1.3 %
Foreign	262	1.1 %
TStyle	494	0.8 %
Magazine	20	0 %
National	3	0 %
Sports	2	0 %

The categories *op-ed*, *science* and *styles* correspond to popularity rates of 77 %, 66 % and 35 % respectively and they can be considered statistically representative in terms of number of blog articles.

In contrast, *business*, *metro*, *culture* and the *undefined* category, which together contain a majority out of all blog articles, range between 17 % and 7 % in popularity rate.

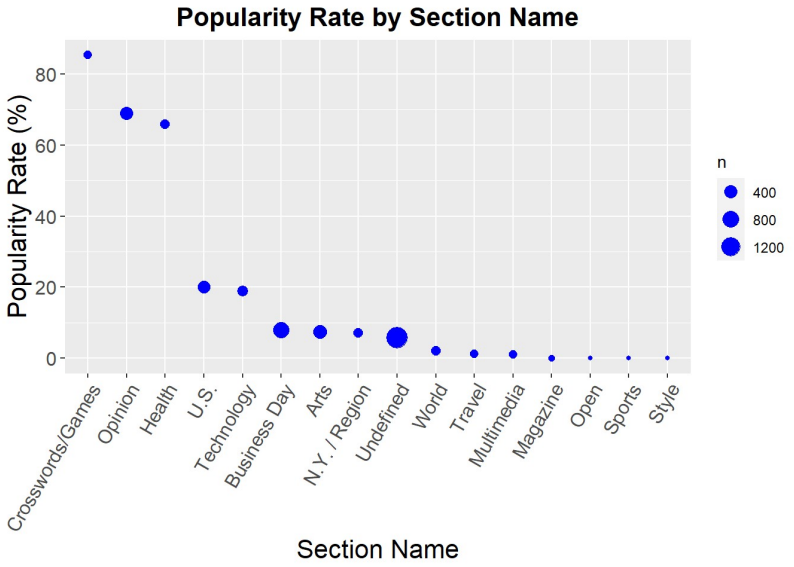
The categories *tstyle*, *travel* and *foreign* have popularity rates of less than 2 %.

The categories *magazine*, *national* and *sports* have a common popularity rate of 0 % but cannot be deemed to be statistically representative in terms of number of blog articles.

In a snapshot, this is a promising insight: even without taking into account the last three categories, *news desks* show some strong and statistically significant differences and they could have substantial predictive power.

It’s time now we turned to the *section name* predictor.

2. *Section Names*



The graph above offers similarity with the graph about *news desks*: three categories have much higher popularity rates than the others. Let's have a look at the corresponding table.

Section Name	Occurrence	Popularity Rate
Crosswords/Games	82	85.4 %
Opinion	402	68.9 %
Health	123	65.9 %
U.S.	339	20.1 %
Technology	201	18.9 %
Business Day	704	8 %
Arts	446	7.4 %
N.Y. / Region	140	7.1 %
Undefined	1 573	5.7 %
World	149	2 %

Section Name	Occurrence	Popularity Rate
Travel	79	1.3 %
Multimedia	92	1.1 %
Magazine	20	0 %
Open	1	0 %
Sports	1	0 %
Style	2	0 %

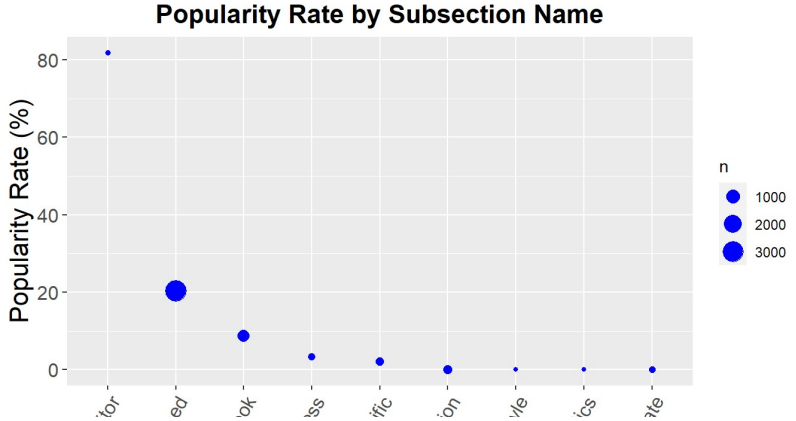
There is a clear-cut difference between some categories: *crosswords/games*, *opinion* and *health* have average popularity rates above 65 %, much higher than the other categories.

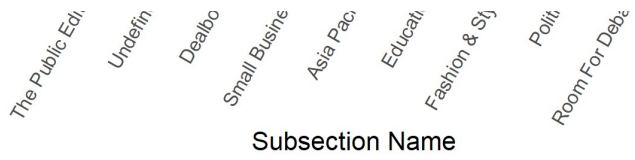
There might be some overlapping between some categories from *news desks* and some categories from *section names*: in the category *science* from *news desks*, there are 125 blog articles with an average popularity rate of 66 % and these statistics are almost the same in the category *health* from *section names*, with 123 blog articles and 66 % average popularity rate…

But there are solid differences as well: among *news desks*, the categories above 65 % total 473 blog articles while, among *section names*, the categories above 65 % total 607 blog articles…

Consequently, without any further investigation, both predictors will be taken on board.

3. *Subsection Names*





In the graph above, there is a very high popularity rate, but how many blog articles correspond to that rate?

Subsection Name	Occurrence	Popularity Rate
The Public Editor	11	81.8 %
Undefined	3 239	20.4 %
Dealbook	612	8.7 %
Small Business	92	3.3 %
Asia Pacific	143	2.1 %
Education	215	0 %
Fashion & Style	2	0 %
Politics	1	0 %
Room For Debate	39	0 %

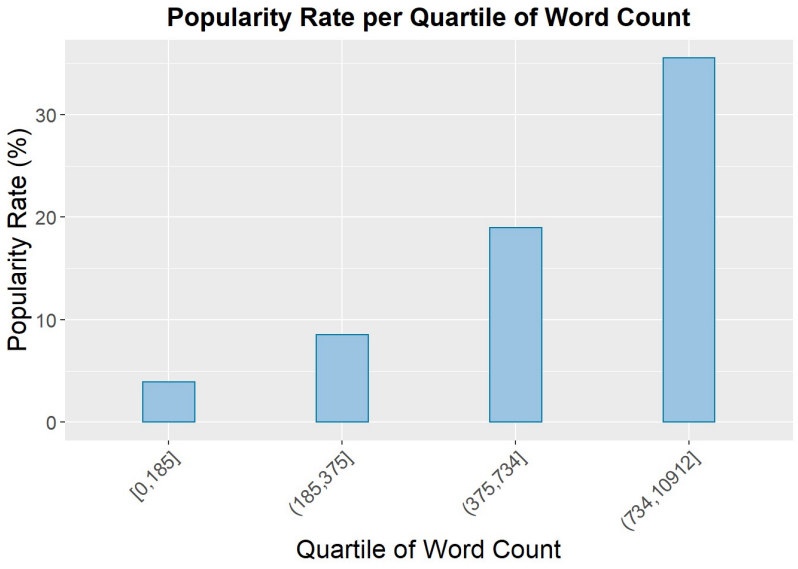
Actually, the popularity rate of 82 % is not really representative from a statistical point of view. But the rate of 20 % is well representative and rather distant from lower rates, and especially from the rates of *small business*, *Asia Pacific* and *education*.

Consequently, this insight makes us accept this predictor as well.

Up to now, three classification factors have been examined among candidate predictors. Pre-attentive insights have been garnered from graphs and precise information has been hoarded from tables. The three classification factors are kept as predictors.

Let’s now examine a numerical predictor: word count.

C. Numerical Predictor: Word Count



The graph above is insightful: measured per quartile of word count, popularity rate is strictly and substantially increasing with word count. Moreover, differences are statistically significant since each average popularity rate refers to approximately one fourth of the training set. Consequently, word count will be used as a predictor in Machine Learning.

We are now ready to move to predicting popularity on the training set, which has been supplemented with

- columns from the Document Term Matrix
- and with two temporal predictors originating from time series decomposition.

VII. MACHINE LEARNING PREDICTION on the TRAINING SET with *EXTREME GRADIENT BOOSTING* and *RANDOM FOREST*

A. Running *eXtreme Gradient Boosting* and *Random Forest* on the Training Set

Based on experience with other projects, several models have been tried on the training set and on bootstrapped resamples, with the package *caret*:

- *Support Vector Machines with Radial Basis Function Kernel* (*svmRadialCost* method),
- *CART* (*rpart* method),
- *Generalized Linear Model* (*glm* method),
- *eXtreme Gradient Boosting* (*xgbLinear* method),
- *Random Forest* (*rf* method).

eXtreme Gradient Boosting and *Random Forest* have emerged as better performing in ROC AUC.

Parameterization has been dealt with in a stepwise approach:

- in a first step, no *tuneGrid* has been provided to the *train()* function from the package *caret* and, by default, the system has picked up ranges on its own;
- the default ranges and the associated results have been perused;
- on the basis of results, new ranges have been stepwise tested on bootstrapped resamples;
- at the end of the process, for each of the two models, one parameter value range has been inserted into the argument *tuneGrid* in the *train()* function hereunder.

By the way, ranges of parameter values have been tested on bootstrapped resamples and not on the whole training set with the final models because the final models were already showing heavy overfitting proneness on the whole training set.

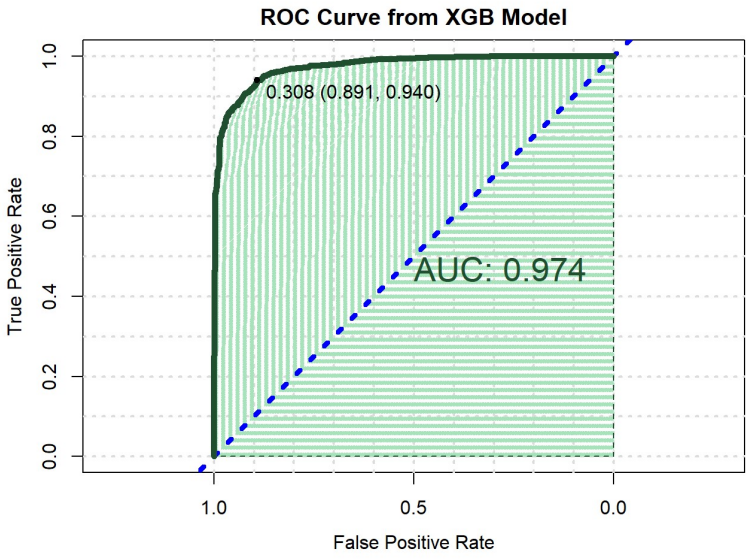
```
fitControl <- trainControl(classProbs = TRUE,
                           ## Evaluating performance using
                           ## the following function.
                           summaryFunction = twoClassSummary,
                           returnResamp = "all",
                           savePredictions = "all")

xgbGrid <- expand.grid(lambda = c(0.70, 0.80, 0.90),
                       alpha = c(0.30, 0.40, 0.50),
                       nrounds = c(26, 27, 28),
                       eta = 0.3)

set.seed(1)
fit_xgb <- train(Popular ~ ., data = train,
                 method = "xgbLinear",
                 trControl = fitControl,
                 ## Asking to tune the model across 3 values
                 ## of the parameters lambda, alpha and nrounds.
                 tuneGrid = xgbGrid,
                 ## Specifying which metric to optimize.
                 metric = "ROC")

rfGrid <- expand.grid(mtry = c(11, 12, 13))

set.seed(1)
fit_rf <- train(Popular ~ ., data = train,
                method = "rf",
                trControl = fitControl,
                ## Asking to tune the model across 3 values
                ## of the parameter mtry.
                tuneGrid = rfGrid,
                ## Specifying which metric to optimize.
                metric = "ROC")

rm(fitControl)
```

Let's compare performance merits from *Random Forest* and *eXtreme Gradient Boosting*. It is only fitting that the first examined objects are the respective ROC curves and ROC AUCs. Let's start with *eXtreme Gradient Boosting*.

B. ROC Curve and AUC from *eXtreme Gradient Boosting*

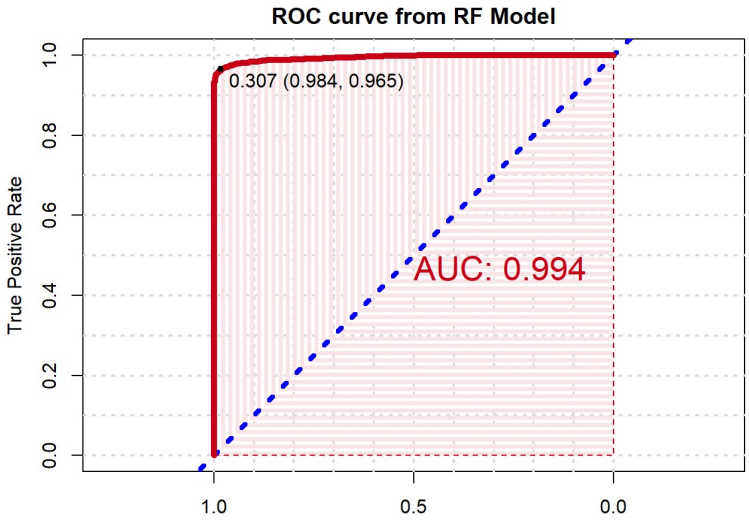
```
fitted_xgb_prob <- predict(fit_xgb, type = "prob")
roc_xgb <- roc(as.factor(train$Popular), fitted_xgb_prob$Popular,
               quiet = TRUE)
```

Here is the graph with the ROC curve and the AUC polygon from *eXtreme Gradient Boosting*.



The true positive rate immediately jumps to 70 % and rather rapidly reaches 100 % to remain there. The AUC is impressive with 97 % but it is calculated on the training set and not on the validation set. Now, let's turn to *Random Forest*.

C. ROC Curve and AUC from *Random Forest*



False Positive Rate

The ROC curve and the AUC are almost perfect, with an AUC of nearly 100 %. But, once again, this is prediction on the training set and not on the validation set.

D. Way Forward

Overfitting is very probable, even in the case of *eXtreme Gradient Boosting*. This makes us ask three questions:

- How could overfitting be estimated?
- Among *eXtreme Gradient Boosting* and *Random Forest*, which is the best model to predict on the validation set?
- Or should we move to another model?

In order to answer these questions, let's make the most of unused information about bootstrapped resample distributions.

VIII. TESTING on BOOTSTRAPPED RESAMPLE DISTRIBUTIONS

A. Preliminary Remark

In the train() function from the package *caret*, the default resampling method has been kept: resampling is done on 25 bootstrapped resamples.

This means that, in the *eXtreme Gradient Boosting* model, there have been 675 AUC estimates. Indeed, there are 27 combinations of parameter values since the *tuneGrid* argument contains 3 values for the parameter *lambda*, 3 for the parameter *alpha* and 3 again for the parameter *nrounds*. And each combination out of the 27 combinations is run in 25 resampling iterations.

In the *Random Forest* model, there have been 75 AUC estimates. Indeed, the *tuneGrid* argument contains 3 values for the parameter *mtry* and each of them is run on 25 bootstrapped resamples.

Looking at all these AUC estimates might deliver insights about overfitting and performance transposability to the validation set.

But before retrieving insights from bootstrapped resample distributions, let's remember that ROC and AUC are somewhat different for *eXtreme Gradient Boosting* and for *Random Forest*. Independently of performance, this gap in internal results might become an insight. Indeed, combining models with different results may better performance… Let's keep that in mind and go back to bootstrapped resample distributions.

B. Insights from All Resample AUCs from *eXtreme Gradient Boosting* and *Random Forest*

Let's get started with a quick summary statistic per model, which is calculated across all AUCs per model, i.e. across 675 AUCs for *eXtreme Gradient Boosting* and 75 AUCs for *Random Forest*. This summary statistic gives

- the minimum of all AUCs per model,
- the mean of all AUCs per model
- and the maximum of all AUCs per model.

	Minimum Resample AUC	Average Resample AUC	Maximum Resample AUC
Random Forest Model	**92.42**	**93.62**	**94.72**
eXtreme Gradient Boosting Model	**92.15**	**93.69**	**95.25**

The table above delivers two insights.

First, AUC levels are substantially lower than the AUC levels of the final models, i.e. 97 % with *eXtreme Gradient Boosting* and almost 100 % with *Random Forest* as shown previously. This insight tends to ascertain the hypothesis of overfitting.

Second, order in performance metric has been partially flipped between both models: *Random Forest*, which delivers an almost perfect AUC of almost 100 % on the final model on the training set, still comes first in minimum resample AUC but is

slightly below in average resample AUC and is half a percentage point below in maximum resample AUC, taking into account all resample AUCs, i.e. 75 for *Random Forest* and 675 for *eXtreme Gradient Boosting*.

C. Insights from AUCs Regrouped per (Set of) Parameter Value(s)

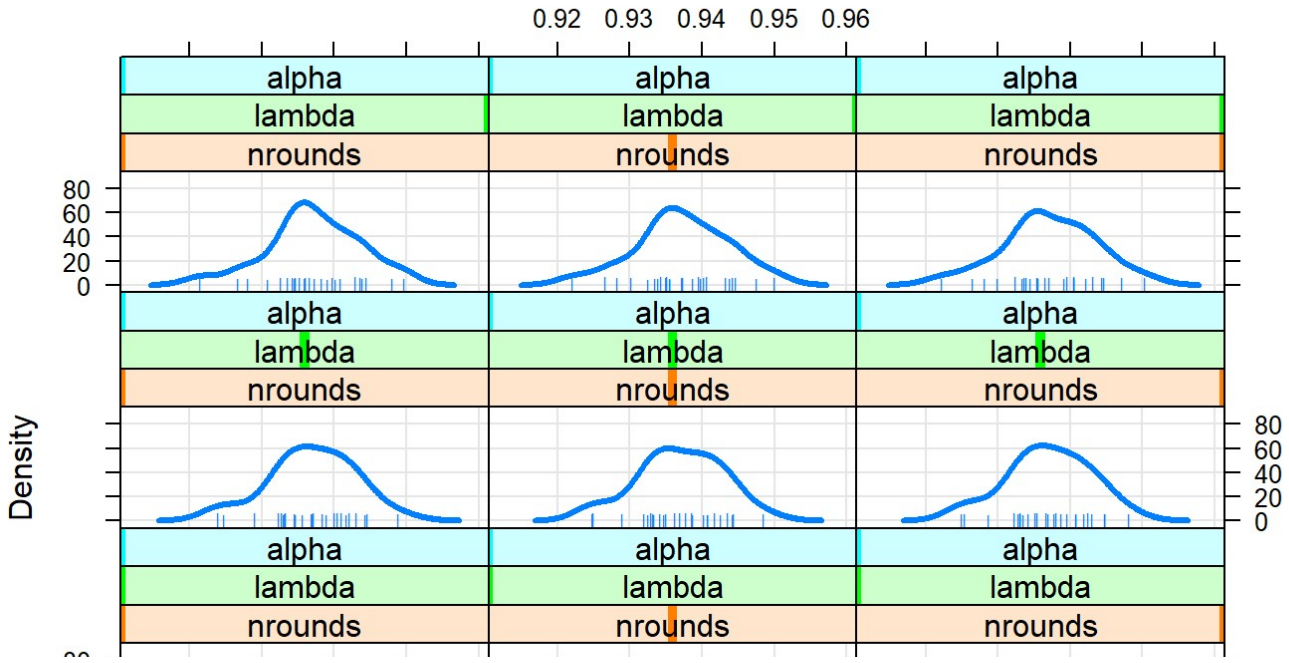
Let's regroup AUCs according to parameterization:

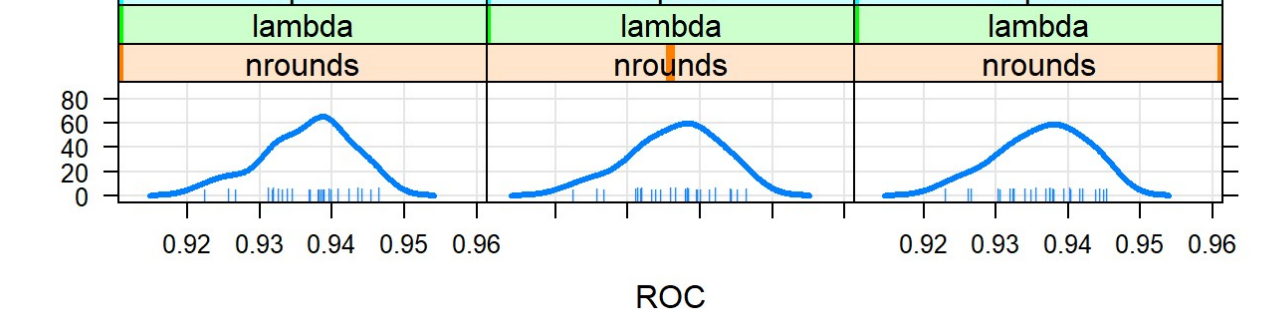
- first, AUCs will be presented in density functions per (set of) parameter value(s);
- second, AUCs will be presented in averages per (set of) parameter value(s).

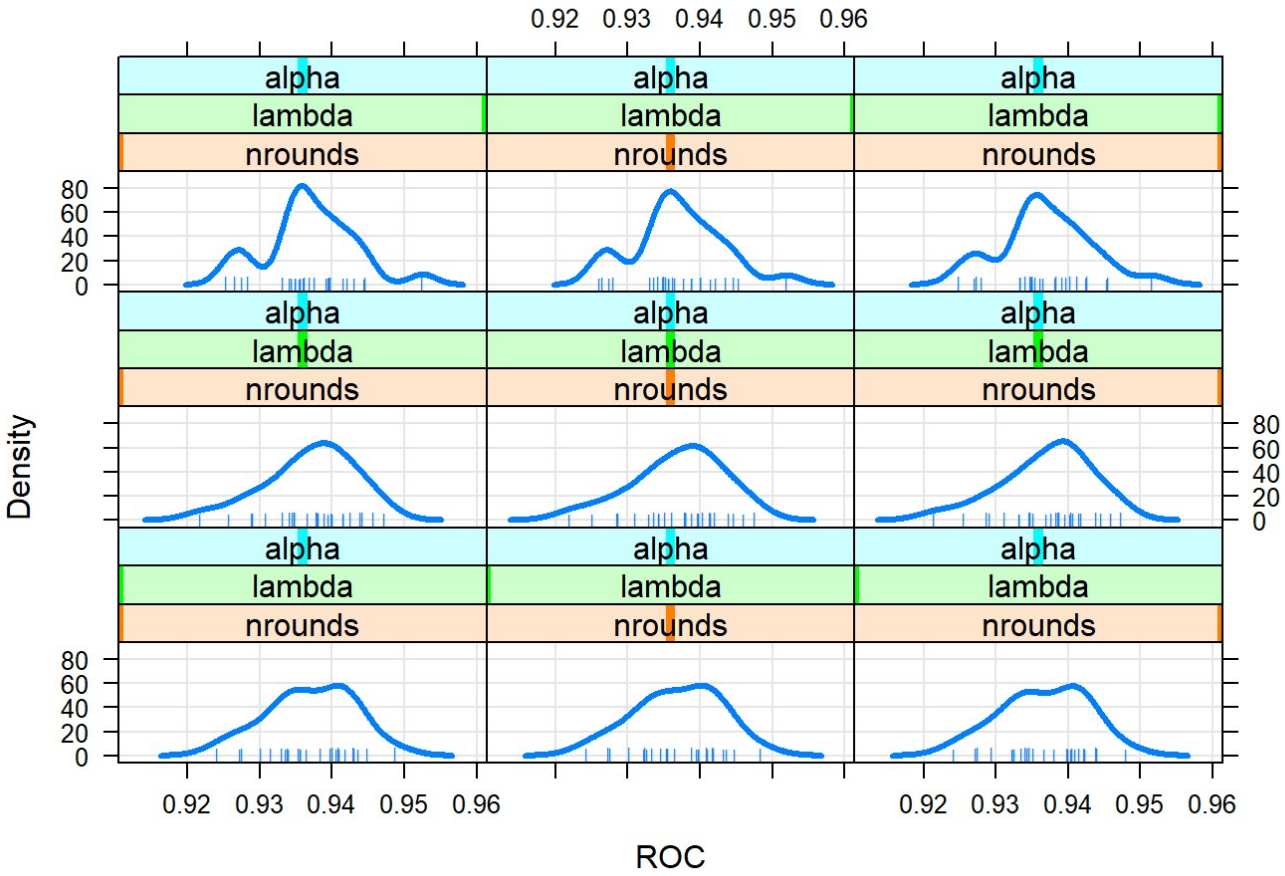
1. AUC Density Functions per (Set of) Parameter Value(s)

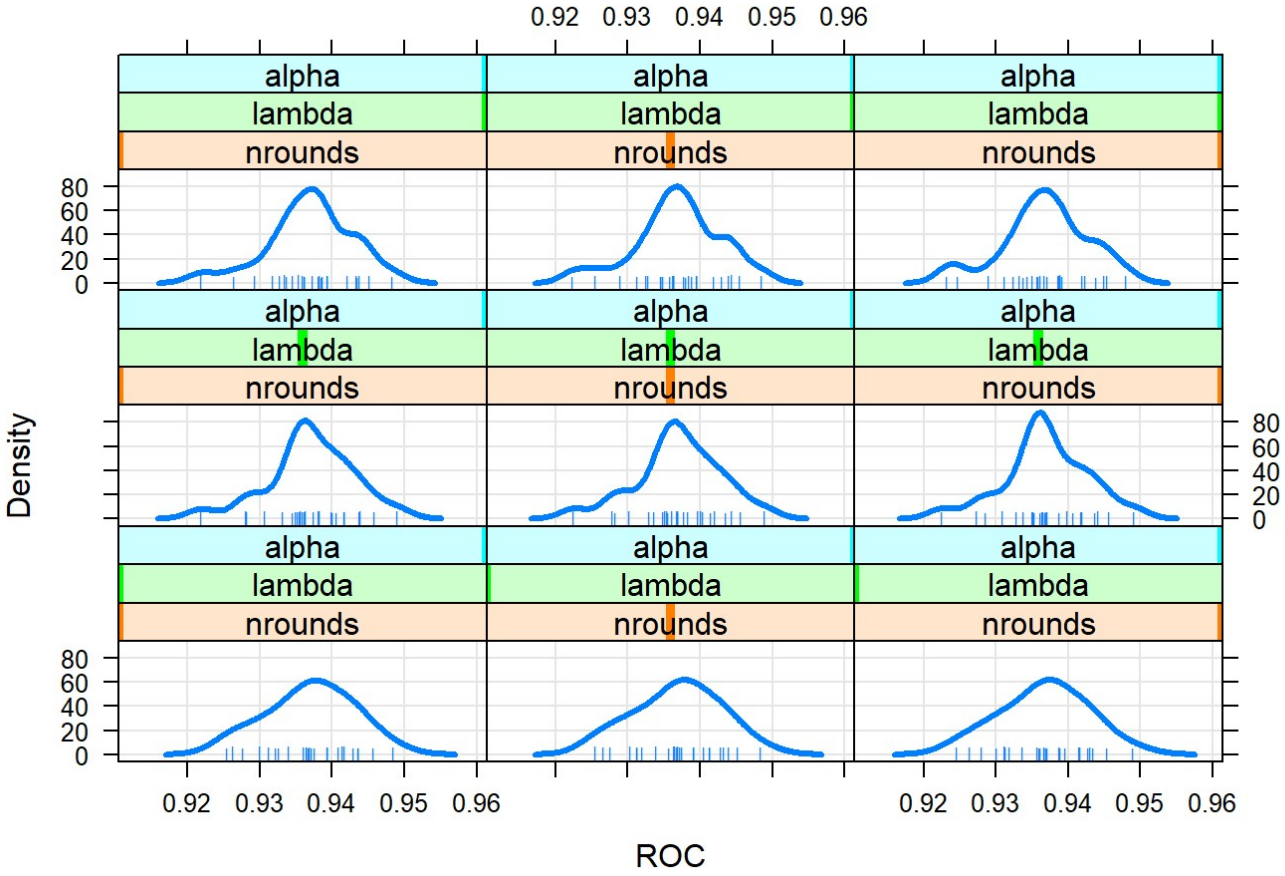
27 density functions will be drawn for *eXtreme Gradient Boosting*, i.e. one per each of the 27 combinations of parameter values; 3 density functions will be produced for *Random Forest*, i.e. one per each of the 3 parameter values tuned on *Random Forest*. Here are the *eXtreme Gradient Boosting* density functions.

```
densityplot(fit_xgb, pch = "|", lwd = 3, grid = T, ylab = "Density")
```



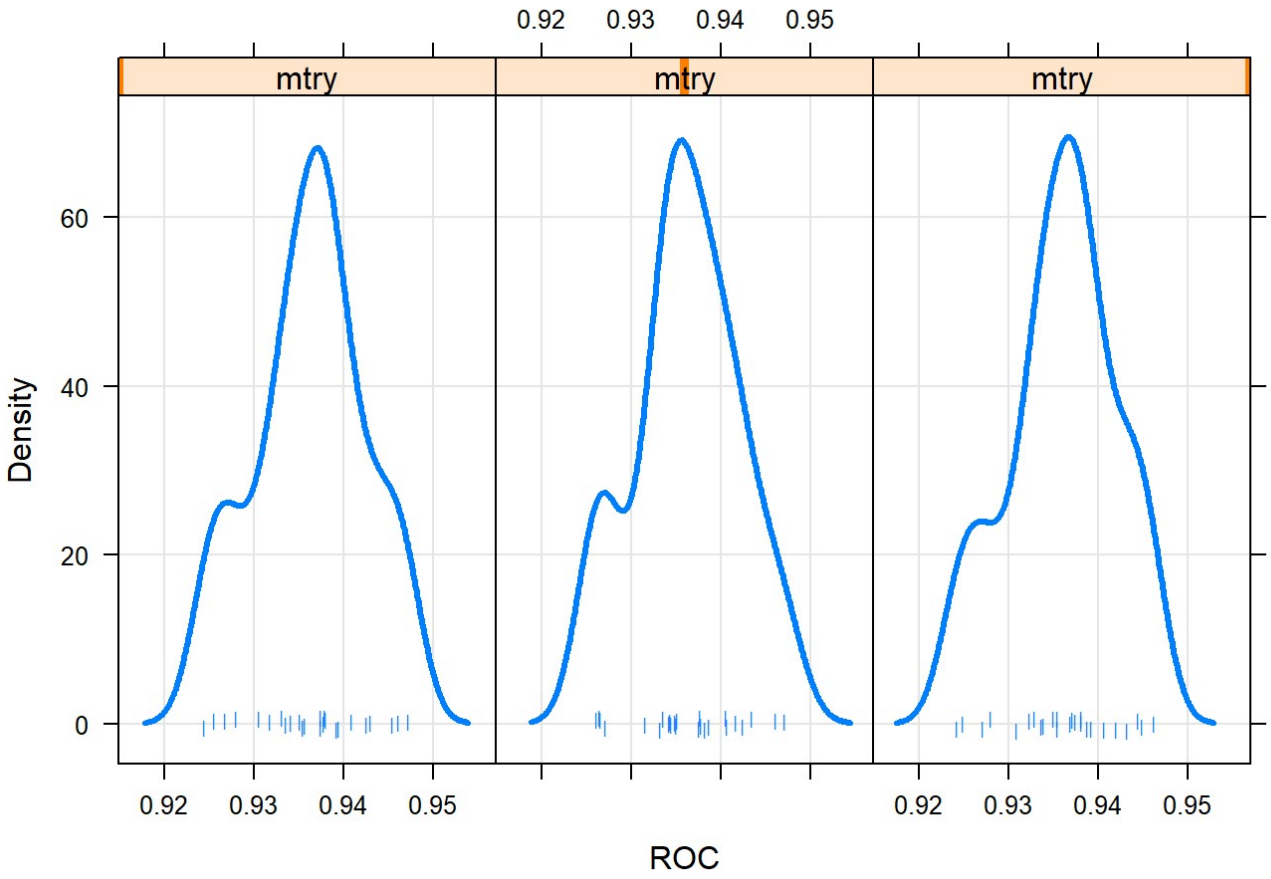

lambda
nrounds
80
60
40
20
0
0.92 0.93 0.94 0.95 0.96
ROC


0.92 0.93 0.94 0.95 0.96
alpha
lambda
nrounds
Density
80
60
40
20
0
ROC


0.92 0.93 0.94 0.95 0.96
alpha
lambda
nrounds
Density
80
60
40
20
0
ROC

The top of the density functions is above 93 % ROC AUC in almost all cases. Let's turn to the *Random Forest* density functions.

```
densityplot(fit_rf, pch = "|", lwd = 3, grid = T, ylab = "Density")
```



The top of the density function is above 93 % in the three cases.

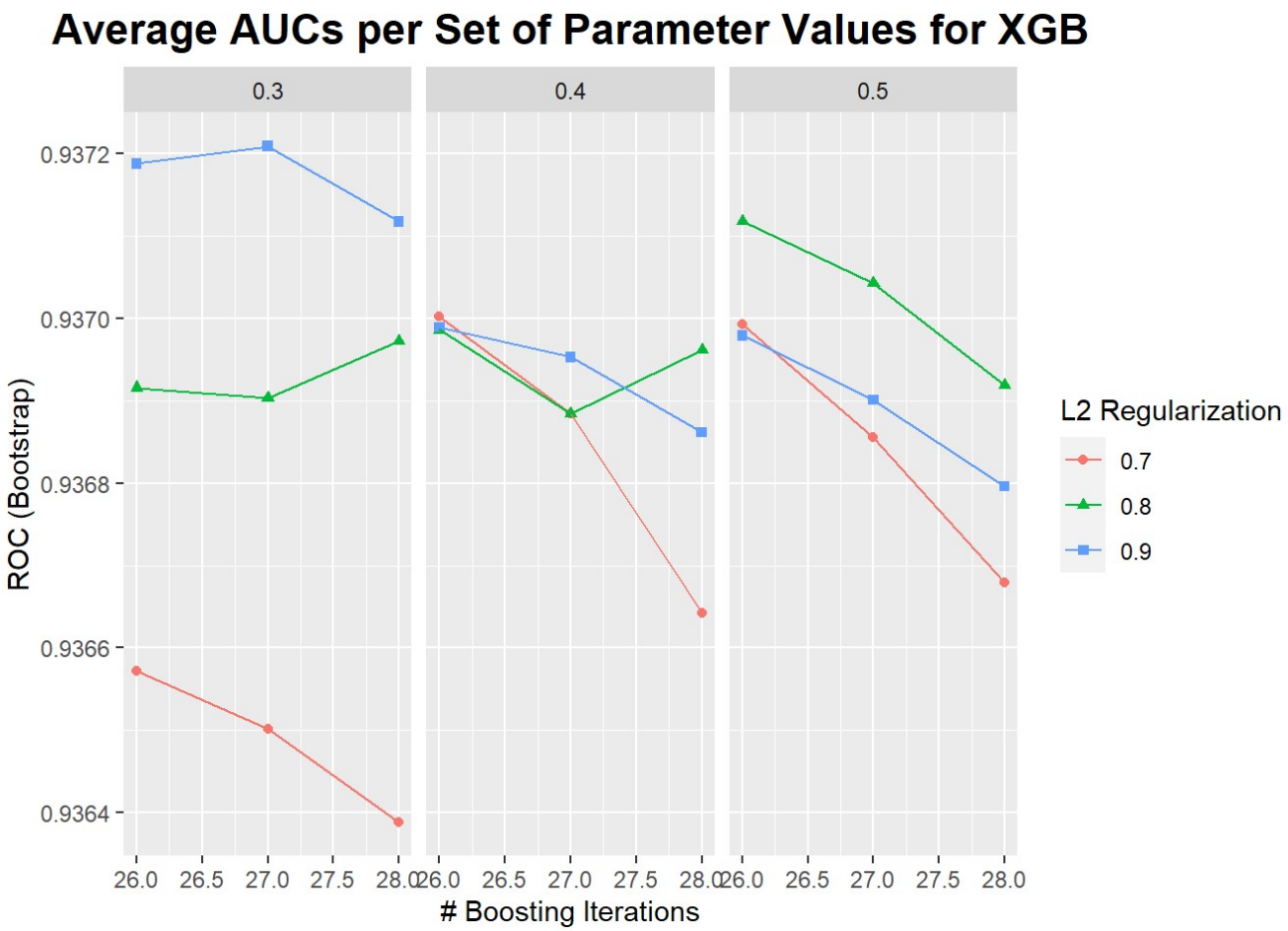
Altogether, both models

- seem very close to each other in AUC performance;
- deliver AUC levels much lower than AUC levels from the final models on the whole training set, which tends to demonstrate overfitting.

2. Average AUCs per (Set of) Parameter Value(s)

For each density function, an average can be calculated over the 25 resamples and corresponds to one (set of) parameter value(s). These averages will be presented as points on graphs, first for *eXtreme Gradient Boosting*.

```
ggplot(fit_xgb) +
  ggtitle("Average AUCs per Set of Parameter Values for XGB") +
  theme(plot.title = element_text(hjust = 0.5, size = 16, face = "bold"))
```



In the graph above

- the upper horizontal “axis” represents the values of the parameter *alpha*;
- the lower horizontal axis represents the values of the parameter *nrounds*;
- the colors represent the parameter *lambda*, whose values are indicated in the legend at the right-hand side of the graph.

From the graph, it can be seen that

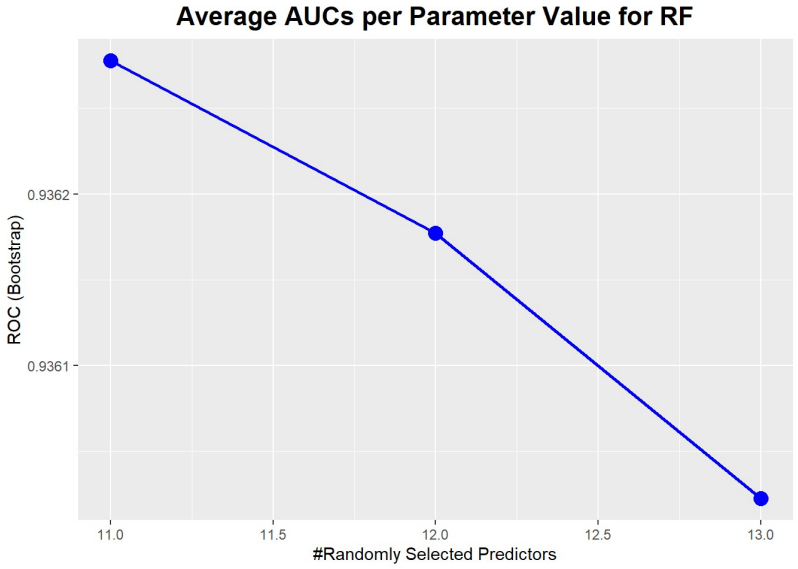
- the highest average is around 93.72 %, which can be considered positive (and much more realistic than the 97.4 % from the final model on the whole training set);
- it corresponds to the parameter values 0.5 for *alpha*, 27 for *nrounds* and 0.9

for *lambda*;

- all averages are in a narrow range, which is reassuring in terms of predictive capability.

Let's do the same for *Random Forest*.

```
ggplot(fit_rf) +
  ggtitle("Average AUCs per Parameter Value for RF") +
  geom_line(col = "blue", size = 1) +
  geom_point(col = "blue", size = 4) +
  theme(plot.title = element_text(hjust = 0.5, size = 16, face = "bold"))
```



The graph for *Random Forest* is much simpler since the *train()* function from the package *caret* tunes the method *rf* only on the parameter *mtry*. Conclusions could read as follows:

- the highest AUC average is slightly below 93.63 %;
- this is slightly below the highest AUC average from *eXtreme Gradient Boosting*;
- the level of highest AUC average and the narrowness of the range are rather promising for *Random Forest*, just as for *eXtreme Gradient Boosting*.

For the record, AUC decreases all the way down the line for *Random Forest* and it might be asked whether it would not further increase for lower values of *mtry*. This has been tested and it would not improve AUC, on the contrary. For precision purposes, the AUC ranges will be digitized in the following table.

```
density_xgb <- fit_xgb$results %>%
  summarise(range = paste(min = round((min(ROC) * 100), 2),
                          max = round((max(ROC) * 100), 2),
                          sep = " - ")) %>%
  select(range)

density_rf <- fit_rf$results %>%
  summarise(range = paste(min = round((min(ROC) * 100), 2),
                          max = round((max(ROC) * 100), 2),
                          sep = " - ")) %>%
  select(range)
```

The ranges are as follows.

	RANGE of AUCs AVERAGED according to PARAMETERIZATION
Random Forest Model	**93.6 - 93.63**
eXtreme Gradient Boosting Model	**93.64 - 93.72**

The *eXtreme Gradient Boosting* range lies very slightly higher than the *Random Forest* range.

For the record, let's check up that the maximum value in each range is indeed the AUC corresponding to the optimal parameter value(s) already pointed to previously and used in the final models on the whole training set.

D. Insights from AUCs Corresponding to Optimal Parameterization

Let's extract, for each model, the AUC corresponding to the optimal parameter value(s).

```
stat <- summary(resamples(list(fit_rf, fit_xgb)))
```

```
## Warning in resamples.default(list(fit_rf, fit_xgb)): 'Model1' did not have
## 'returnResamp="final"; the optimal tuning parameters are used
```

```
## Warning in resamples.default(list(fit_rf, fit_xgb)): 'Model2' did not have
## 'returnResamp="final"; the optimal tuning parameters are used
```

```
tab <- round((stat$statistics$ROC[7:8] * 100), 2)
rm(stat)
```

Deliberately, warnings have been allowed to get confirmation that we are indeed getting the AUCs corresponding to optimal parameterization. For the record, verification could also have been performed e.g. by using *fit_xgb$bestTune* and *fit_rf$bestTune* and applying them to retrieve the corresponding average AUCs from *fit_xgb*$results$*ROC* and *fit_rf*$results$*ROC*… Here is the comparative table with the AUCs, which are indeed the range maxima from the previous table.

	RESAMPLE AUC with OPTIMAL PARAMETERIZATION
Random Forest Model	**93.63**
eXtreme Gradient Boosting Model	**93.72**

eXtreme Gradient Boosting scores very slightly higher than *Random Forest*. Both are around 93.60 / 93.70 %. This level is an indicator of predictive capability by both models on the validation set.

AUCs on the validation set would most probably be much closer to this 93.60 / 93.70 % level than to the levels previously delivered by both final models on the whole training set, i.e. respectively 97 % and almost 100 %…

E. Taking Stock of Insights from Bootstrapped Resamples

The table above is the last step in our evaluating the predictive power of the

eXtreme Gradient Boosting and *Random Forest* models thanks to bootstrapped resample information.

First, AUCs have been collected on the 675 resamples from *eXtreme Gradient Boosting* and the 75 resamples from *Random Forest*. Second, AUCs have been regrouped according to parameterization, with 27 parameter value combinations for *eXtreme Gradient Boosting* and 3 parameter values for *Random Forest*. Third, from these regroupments only AUCs corresponding to optimal parameterization have been kept in the table above. In the three steps, *Random Forest* has proved a very little bit higher in AUC.

But AUC has dramatically dropped with respect to the AUCs from the final models on the training set. Instead of being at 97 or almost 100 %, AUC is now around 93.60 / 93.70 % on resamples and this lower level is most probably closer to predictive capability on the validation set. This drop in AUC looks like a stimulus to further modelling towards AUC optimization.

To face this challenge, it is time to remember an insight that had been previously garnered: output differences from *eXtreme Gradient Boosting* and *Random Forest* can be an insight towards combining models into an ensemble solution.

F. TESTING an ENSEMBLE SOLUTION on BOOTSRAPPED RESAMPLE DISTRIBUTIONS

An ensemble solution will be built by combining probabilities of the positive class originating from the two individual models: for each blog article, popularity will be predicted using the average of the probabilities produced respectively by *eXtreme Gradient Boosting* and *Random Forest*.

Before opting into validating the ensemble solution on the validation set, the predictive power of the ensemble solution will be tested on the bootstrapped resamples, focusing only on optimal parameterization results.

```
# Retrieving optimal parameter values.
bt_xgb <- fit_xgb$bestTune
bt_rf <- fit_rf$bestTune

# Getting probabilities of the positive class with optimal parameterization and ave
rage on the 25 bootstrapped resamples.
df_xgb <- fit_xgb$pred %>%
          as.data.frame(stringsAsFactors = FALSE) %>%
          filter(lambda == bt_xgb$lambda,
                 alpha == bt_xgb$alpha,
                 nrounds == bt_xgb$nrounds) %>%
          group_by(rowIndex) %>%
          summarize(prob_popular = mean(Popular)) %>%
          select(rowIndex, prob_popular)

df_rf <- fit_rf$pred %>%
         as.data.frame(stringsAsFactors = FALSE) %>%
         filter(mtry == bt_rf$mtry) %>%
         group_by(rowIndex) %>%
         summarize(prob_popular = mean(Popular)) %>%
         select(rowIndex, prob_popular)

# Calculating the mean of XGB probability and RF probability for each observation.
prob_popular_ensemble <- (df_xgb$prob_popular + df_rf$prob_popular) / 2

# ROC function of the ensemble solution
ro <- roc(as.factor(train$Popular), prob_popular_ensemble)

# AUC of the ensemble solution
auc_ensemble <- round((auc(ro) * 100), 2)
```

Let's have a look at the AUC produced by the ensemble solution, compared with the corresponding AUCs from *eXtreme Gradient Boosting* and *Random Forest*.

	RESAMPLE AUC with OPTIMAL PARAMETERIZATION
Random Forest Model	**93.63**
eXtreme Gradient Boosting Model	**93.72**
Ensemble Solution XGB + RF	**94.12**

G. Partial Conclusion from Bootstrapping Information

On the basis of bootstrapped resamples, the ensemble solution shows some AUC upgrade. Limited though the upgrade is, this insight is deemed to be statistically significant since it originates from stepwise and extensive analysis of bootstrapping iterations.

The ensemble solution will be tentatively validated on the validation set. It will deliver blog article popularity predictions, based, by definition, on predictions by both individual models.

IX. VALIDATION on the VALIDATION SET

A. Constructing the Validation Set

The training set has been built up completely separately, to avoid any interaction with the validation set, which will be constructed now.

```
# Replacing PubDate by two components just as in the training set.
valNews <- valNews %>%
  mutate(date = as.POSIXct(PubDate)) %>%
  mutate(weekday = weekdays(date, abbreviate = TRUE)) %>%
  mutate(hour = hour(date)) %>%
  as.data.frame(stringsAsFactors = FALSE) %>%
  select(- PubDate, - date, - Abstract, - UniqueID)

# Combining headlines and snippets.
v <- 1:nrow(valNews)
for (i in 1:nrow(valNews)) {
  v[i] <- paste(valNews$Headline[[i]],
                valNews$Snippet[[i]], sep = " ")
}

# Corpus is created on validation reviews only to avoid any interference
# between training reviews and validation reviews. Otherwise,
# tokens from validation set could have (slightly) impacted on token selection
# when applying the sparsity threshold.
corpus <- VCorpus(VectorSource(v))

# Lowercasing, removing punctuation and stopwords, stemming document.
corpus <- tm_map(corpus, content_transformer(tolower))
corpus <- tm_map(corpus, removePunctuation)
corpus <- tm_map(corpus, removeWords, stopwords("english"))
corpus <- tm_map(corpus, stemDocument)

# Building up a bag of words in a Document Term Matrix.
dtm <- DocumentTermMatrix(corpus)

# Managing sparsity with sparsity threshold.
sparse <- removeSparseTerms(dtm, 0.975)

# Converting sparse, which is a DocumentTermMatrix,
# to a matrix and then to a data frame.
sentSparse <- as.data.frame(as.matrix(sparse))

# Making all column names R-friendly.
colnames(sentSparse) <- make.names(colnames(sentSparse))

# For machine learning, columns have to match between training set
# and validation set: adjustments have to be made on the validation set.

# Let's keep only columns that also exist in the training set.
sentSparse <- sentSparse %>% as.data.frame() %>%
  select(intersect(colnames(.), colnames(sentSparse_train)))

# Columns from train that are missing in val have to be added as null vectors.
mis <- setdiff(colnames(sentSparse_train), colnames(sentSparse))
df <- data.frame(matrix((nrow(sentSparse) * length(mis)),
                 nrow = nrow(sentSparse), ncol = length(mis)) * 0) %>%
      `colnames<-`(mis)
buffer <- cbind(sentSparse, df) %>% as.data.frame()
```

```
val <- cbind(valNews, buffer) %>% select(- Headline, - Snippet)

rm(corpus, dtm, sparse, sentSparse, sentSparse_train, valNews)
rm(v, i, mis, df, buffer)
```

B. Prediction on the Validation Set

```
# On the validation set, predicting with XGB, calculating ROC curve and AUC.
pred_xgb <- predict(fit_xgb, newdata = val, type = "prob")
roc_xgb <- roc(as.factor(val$Popular), pred_xgb$Popular)
auc_xgb <- round((auc(roc_xgb) * 100), 2)

# On the validation set, predicting with RF, calculating ROC curve and AUC.
pred_rf <- predict(fit_rf, newdata = val, type = "prob")
roc_rf <- roc(as.factor(val$Popular), pred_rf$Popular)
auc_rf <- round((auc(roc_rf) * 100), 2)

# On the validation set, predicting with the ensemble solution, calculating ROC cur
ve and AUC.
hope <- data.frame(Popular = (pred_xgb$Popular + pred_rf$Popular) / 2)
roc_ens <- roc(as.factor(val$Popular), hope$Popular)
auc_ens <- round((auc(roc_ens) * 100), 2)
```

C. Final AUCs

The following table shows the AUC for *eXtreme Gradient Boosting*, *Random Forest* and the ensemble solution.

	AUC on the VALIDATION SET
Random Forest Model	**93.98**
eXtreme Gradient Boosting Model	**94.07**
Ensemble Solution XGB + RF	**94.33**

The table above indicates that

- AUCs on the validation set are practically the same as average AUCs

calculated with optimal parameterization on the bootstrapped resamples from the training set;
- the ensemble solution reaches a very positive AUC level of 94.33 %, closely followed by *eXtreme Gradient Boosting* and *Random Forest*;
- *eXtreme Gradient Boosting* performs slightly better than *Random Forest*, in line with bootstrapping results and contrary to results on the whole training set, where *Random Forest* showed more presumption of overfitting with almost 100 % AUC.

X. CONCLUSION

Natural Language Processing on textual predictors has produced predictors that have been combined with categorical and numerical predictors, such as timing, classifications and word count.

Together they have formed the datasets on which *eXtreme Gradient Boosting* and *Random Forest* have been run.

On the training set, both models were overfitting, especially *Random Forest*. Working on bootstrapped resample distributions has clearly circumscribed the real capabilities of both models and has made it possible for an ensemble solution to be tested in a realistic way.

In predicting on the validation set, the individual models *eXtreme Gradient Boosting* and *Random Forest* generate AUC levels of the ROC curve around 94 %, *Random Forest* scoring a little bit higher.

The ensemble solution combining *eXtreme Gradient Boosting* and *Random Forest* reaches 94.33 % in AUC, which can be considered a solid result.

XI. REFERENCES

A. News Popularity Prediction

https://www.researchgate.net/publication /306061597_Predicting_the_Popularity_of_News_Articles

file:///C:/Users/Acer/Downloads/PAA_ModellingAndPredictingNewsPopularity_13112012.pdf

file:///C:/Users/Acer/Downloads/4646-21907-1-PB.pdf

https://minimaxir.com/2017/06/reddit-deep-learning/

https://medium.com/@syedsadiqalinaqvi/predicting-popularity-of-online-news-articles-a-data-scientists-report-fac298466e7

B. ROC Curves and AUC

https://topepo.github.io/caret/model-training-and-tuning.html#alternate-performance-metrics

https://datascience.stackexchange.com/questions/806/advantages-of-auc-vs-standard-accuracy

Xavier Robin, Natacha Turck, Alexandre Hainard, Natalia Tiberti, Frédérique Lisacek, Jean-Charles Sanchez and Markus Müller (2011). “pROC: an open-source package for R and S+ to analyze and compare ROC curves”. BMC Bioinformatics, 12, p. 77. DOI: 10.1186/1471-2105-12-77

https://cran.r-project.org/web/packages/pROC/pROC.pdf

https://stackoverflow.com/questions/30366143/how-to-compute-roc-and-auc-under-roc-after-training-using-caret-in-r

https://www.rdocumentation.org/packages/pROC/versions/1.16.2/topics/roc

https://www.rdocumentation.org/packages/pROC/versions/1.16.2/topics/plot.roc

C. Resampling and Distributions

https://rafalab.github.io/dsbook/machine-learning-in-practice.html#exercises-55

https://www.rdocumentation.org/packages/caret/versions/6.0-86/topics/trainControl

https://www.rdocumentation.org/packages/lattice/versions/0.3-1/topics

/densityplot