TapanYunusEmre.DA5030.Project

Yunus Emre Tapan

4/13/2022

Contents

1 Signature Project Requirements

1. Identify a data set that you want to explore and for which you can build a minimum of three appropriate and useful machine learning or data mining models.
2. Your effort must follow the CRISP-DM process and addresses business understanding, definition of the problem to be solved, data sources, data cleaning efforts, assessment of data quality, exploration of the data, transformations, imputation, case elimination, training vs validation data set division strategy, model selection, model tuning, evaluation, accuracy, etc. This resource summarizes many of the techniques (Links to an external site.) and can be helpful.
3. Explain in detail what you did, show your R code in an R Notebook, and explain why you chose a particular approach versus other possible approaches. Use proper journaling methods so we know what you did, what your thinking was, and why you chose a particular approach.
4. Your models must be evaluated and compared using appropriate methods, e.g., MAD, MSE, RMSE, accuracy, AUC, R-Squared, etc.
5. You must construct an ensemble learner from your models and also evaluate the ensemble.
6. You must provide justification, interpretation, evaluation, evidence for all your work. Don't simply provide the R output - you must interpret the output and evaluate it.

1.1 Self Rubric Form

Link

2 Proposal

In my signature project, I would like to work on Twitter Data. Twitter data can be retrieved via its API (https://developer.twitter.com/en/docs/twitter-api). I have an Academic Twitter API access and I gathered a couple of millions of tweets based on keywords I determined related to my research question. After some filtrations, my final dataset includes 640618 examples and 6 variables. In general, I try to understand how Turkish-speaking Twitter users speak about the United States in the last decade. Toward this aim, I would like to build a classification model to predict the polarization sentiment of Tweets in terms of being positive or negative. I found a Turkish sentiment dictionary and I will get sentiment scores of tweets to be used in my models. However, I found that I need to create a further cleaning procedure for my dataset to eliminate unrelated tweets. For that part, I aim to get a sample from the dataset and label unnecessary and related tweets. Then, I will build Naive Bayes classification, Support Vector Machines, kNN clustering, and Random Forest models by using my training data. As I have text data to be classified, I found these four ML models very appropriate in my case. When it comes to evaluating my models, I will use precision, recall, and accuracy measurements obtained from the confusion matrix. Apart from supervised machine learning methods, I would like to summarize the tweet data by unsupervised topic modeling approaches though it does not have any evaluation method. This project uses a unique dataset collected two weeks ago by myself(which will be labeled manually) and employs and evaluates multiple classification algorithms to detect unrelated tweets and measure polarization sentiment among tweets.

```r
library(ggplot2)
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(quanteda)
```

```
## Package version: 3.2.1
## Unicode version: 13.0
## ICU version: 67.1
```

```
## Parallel computing: 8 of 8 threads used.
```

```
## See https://quanteda.io for tutorials and examples.
```

```r
library(e1071)
library(caret)
```

```
## Loading required package: lattice
```

```r
library(class)
```

3 Data Retrieval, Exploration and Preprocessing

Tutorial

3.1 Load Data

where does the data come from?

```
wh <- read.csv("https://raw.githubusercontent.com/tapanyemre/Tweet-Classification/main/Data/sample_WH.cs
at <- read.csv("https://raw.githubusercontent.com/tapanyemre/Tweet-Classification/main/Data/sample_Atlan
dc <- read.csv("https://raw.githubusercontent.com/tapanyemre/Tweet-Classification/main/Data/sample_DC.cs
data <- rbind(wh, at, dc)
head(data)
```

No	text	label
1	İyi geceler (@ Beyaz Saray Konakları in Gaziantep) https://t.co/lg8VzAk8wr	0
2	Washington DC'de şu anda saat 22:30. Beyaz Saray önünde çok az sayıda gösterici var. https://t.co/KrOvH8lKWd	1
3	ABD' koronavirüs yası! Bayraklar yarıya indiriliyor Beyaz Saray Sözcüsü Jen Psaki, 500 bin Amerikalının Covid-19 sonucu ölümü nedeniyle Başkan Joe Biden'ın tüm federal binalardaki bay… https://t.co/jkC1kOMbWS	1
4	Pablo Escobar ın üşümesin diye uğruna 128 milyar dolar yaktığı çocuğuyla beyaz saray anısı…	

Binali Yıldırım Erkan Yıldırım #Hükümetİstifa Tayyip #sedatpeker7 Mehmet Ağar https://t.co/LWyEwhUJ4S | 0| | 5|Beyaz Saray: Suriye'ye yönelik yeni saldırılar olabilir https://t.co/wRkl3iQgyj | 1| | 6|İyi geceler olsun canlar.. (@ Beyaz Saray Konakları in Gaziantep) https://t.co/TD7uW3kMFG | 0| My sample dataset includes three columns in terms of and 3000 observations. I extracted three samples from three tweet dataset I collected for my sentiment analysis project. Then, I labeled the tweets as 0 (out of topic) and 1 (relevant). I saw a pattern among irrelevant tweets among tweeets collected for different keywords so that I thought using classifiers would be a good idea for data cleaning process. In this project, I will employ multiple classification algorithms to detect unrelated tweets.

```
table(data$label)
```

```
##
##    0    1
##  623 2377
```

In my dataset, I have 623 unrelated tweets and 2377 relevant tweets.

3.2 Check Data Quality and Missing Values

how do you plan on assessing data quality and deal with missing values?

Let's check whether we have any missing values:

```
length(which(!complete.cases(data)))
```

```
## [1] 0
```

Great, I do not have any missing values.

what strategy are you using for data imputation and why? if the data set has no missing data, can you randomly remove data and then impute the data and compare performance of your algorithms with imputed vs full data? why do they differ? how do they differ?

Although our dataset is complete, assume that our first one hundred label values are missing. In this case, my strategy would be assigning 1 (as the mode of the dataset) to the missing values.

```
data_imputed <- data
data_imputed$label[1:100] <- 1
table(data_imputed$label)
```

```
##
##    0    1
##  608 2392
```

In this case, the number of relevant tweets in our dataset has been increased compared to the original dataset.

3.3 Assess Normality

how do you assess normality, distribution, skew – and does it matter for your algorithms?

As we explored before, our distribution is like below:

```
#checking if the data is balanced
prop.table(table(data$label))
```

```
##
##         0         1
## 0.2076667 0.7923333
```

```
prop.table(table(data_imputed$label))
```

```
##
##         0         1
## 0.2026667 0.7973333
```

I think this distribution is enough to avoid class imbalance like undersampling and oversampling.

3.4 Exploratory Visualizations

what kinds of exploratory data analysis and visualization do you plan on doing?

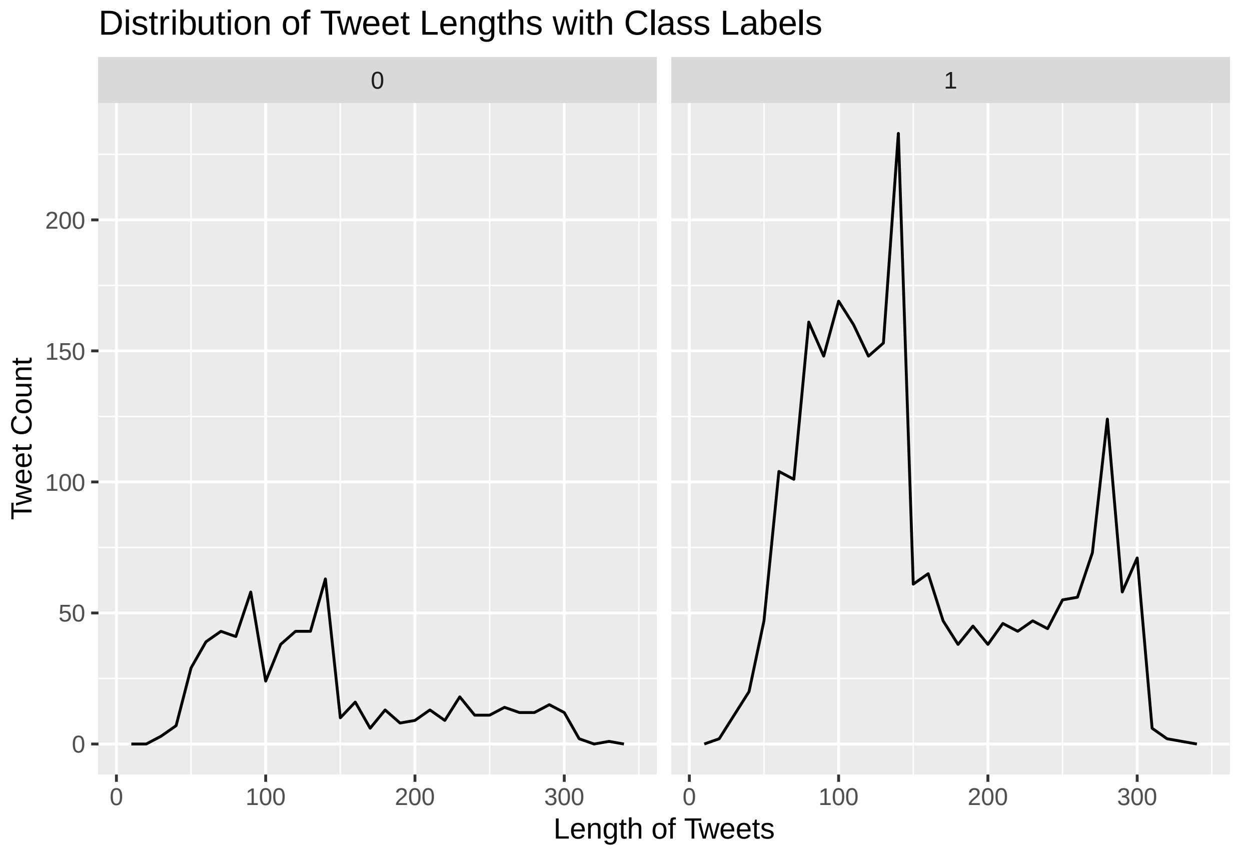
First, let's get a feel for the distribution of text lengths of tweets by adding a new feature for the length of each tweet.

```
data$length <- nchar(data$text)
data_imputed$length <- nchar(data_imputed$text)
summary(data$length)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    24.0    93.0   131.0   150.0   206.2   335.0
```

Let's visualize distribution with ggplot2 by adding segmentation for relevant/unrelevant.

```
ggplot(data, aes(x = length)) +
  facet_wrap(~label)  +
  geom_freqpoly(binwidth = 10) +
  labs(y = "Tweet Count", x = "Length of Tweets",
       title = "Distribution of Tweet Lengths with Class Labels")
```

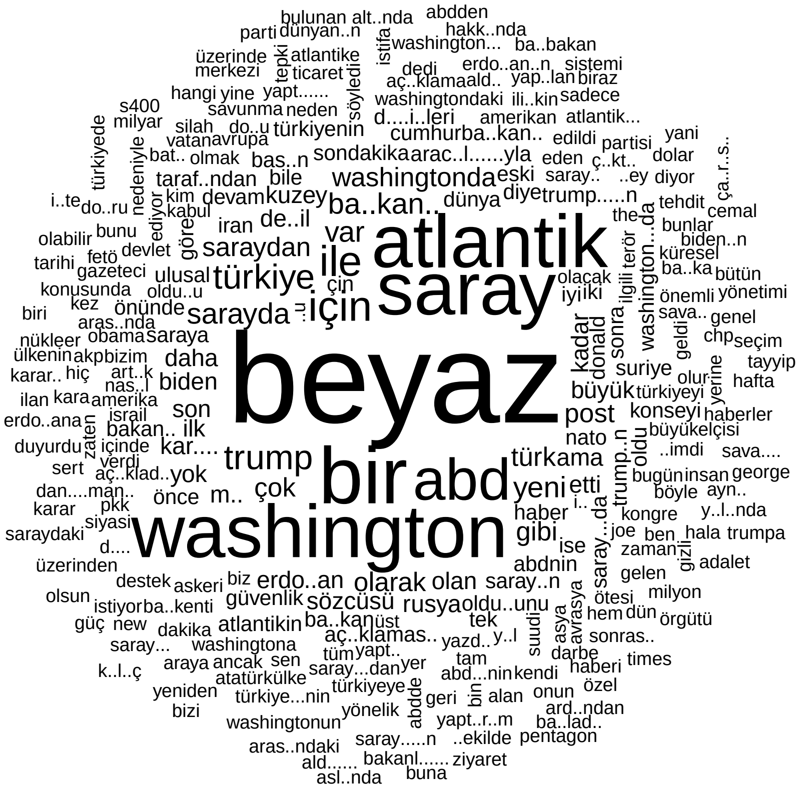


It looks relevant tweets are more lengthy than unrelevant tweets but there is no sharp difference in terms of distribution below 140 characters. This makes sense as tweets were limited to 140 characters before.

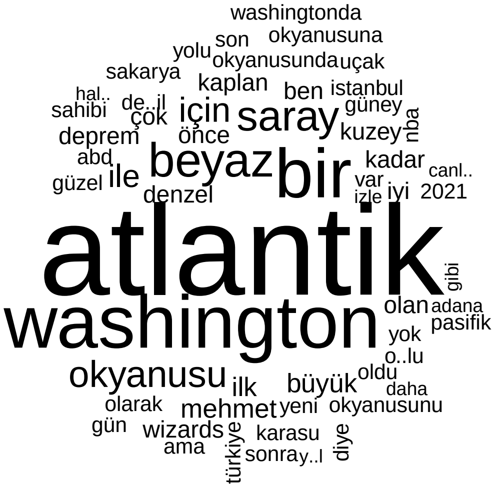
Second, let's look at wordcloud distribution:

```
#visualizing  with word cloud of the words with more than 15,10 and20 frequency
tweets_relevant <- data %>% filter(label==1)
tweets_irrelevant <- data %>% filter(label==0)
wordcloud(tweets_relevant$text, min.freq =15, random.order = FALSE)
```

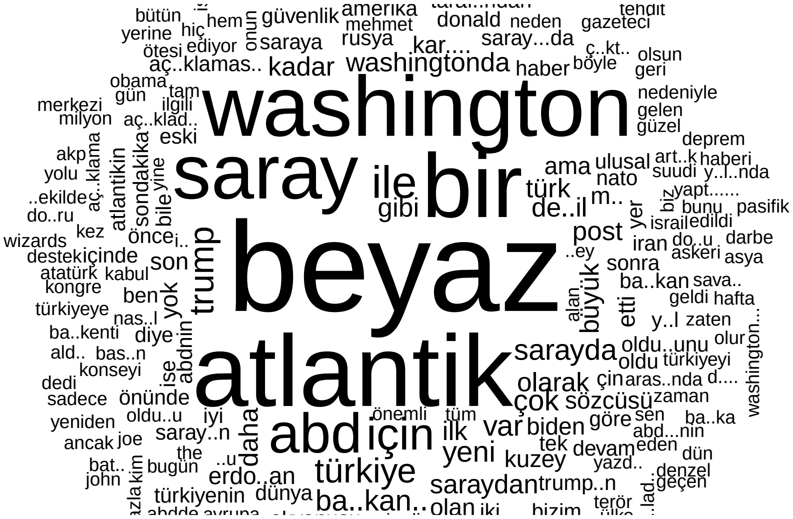
```
## Loading required namespace: tm
```



```
wordcloud(tweets_irrelevant$text, min.freq =10, random.order = FALSE)
```



```
wordcloud(data$text, min.freq =20, random.order = FALSE)
```



Worcloud visualizations shows the sharp difference between relevant and irrelevant tweets. Most frequent words among irrelevant tweets is not about politics which is our main focus here.

3.5 Split Datasets

How do you choose training vs validation data? why? Can you and should you use k-fold cross-validation?

I will use sample.int function to get random sample of row number for training dataset. I will get validation dataset via excluding training rows.

```
# Set the random seed for reproducibility.
set.seed(12345)
#use sample function to get random row numbers for the split
trainRows <- sample.int(n = nrow(data),
                        size = floor(.70*nrow(data)), replace = F)
train <- data[trainRows,2:3]
test <- data[-trainRows,2:3]

# Use caret to create stratified folds for 3-fold cross validation repeated
# 3 times (i.e., create 9 random stratified samples)
cv.folds <- createMultiFolds(train$label, k = 3, times = 3)
cv.cntrl <- trainControl(method = "repeatedcv", number = 3,
                         repeats = 3, index = cv.folds)
```

```
labels_train <- data[trainRows,3]
labels_test <- data[-trainRows,3]
prop.table(table(labels_train))
```

```
## labels_train
##         0         1
## 0.2085714 0.7914286
```

```
prop.table(table(labels_test))
```

```
## labels_test
##         0         1
## 0.2055556 0.7944444
```

3.6 Data Preprocessing and Feature Selection

Feature Selection Tutorial

How will you select the features? will you use PCA? What kind of feature engineering will you use? will you add new derived features? What do you plan on predicting?

what kind of normalization, standardization, regularization, or transformation do you plan on using and why?

I will apply standard text cleaning pipeline here. Basically, the process includes converting all words to lowercase, removing numbers, removing stopwords, removing punctuations, stemming and removing white space. At the end, we will transform our dataset into document-term matrix to make it suitable for classification models. Every rows in the document-term matrix represent tweet's attributes. The final output have numbers in the observations and words in the columns

```
# Tokenize tweet texts
train.tokens <- tokens(train$text, what = "word",
                       remove_numbers = TRUE, remove_punct = TRUE,
                       remove_symbols = TRUE)
# Take a look at a specific tweet and see how it transforms.
train.tokens[[123]]
```

```
##  [1] "Ettiğin"                   "lafa"
##  [3] "kendinin"                  "de"
##  [5] "inandığına"                "inanmıyorum"
##  [7] "Tamamen"                   "atlantik"
##  [9] "ağzı"                      "ABD"
## [11] "dış"                       "ilişkiler"
## [13] "görevlisi"                 "misin"
## [15] "mübarek"                   "https://t.co/6E8JJc6SYh"
```

```
# Lower case the tokens.
train.tokens <- tokens_tolower(train.tokens)
train.tokens[[123]]
```

```
##  [1] "ettiğin"                   "lafa"
##  [3] "kendinin"                  "de"
##  [5] "inandığına"                "inanmıyorum"
##  [7] "tamamen"                   "atlantik"
##  [9] "ağzı"                      "abd"
## [11] "dış"                       "ilişkiler"
## [13] "görevlisi"                 "misin"
## [15] "mübarek"                   "https://t.co/6e8jjc6syh"
```

```
# Use a specific stopword list from stopwords-iso dictionary for Turkish text
train.tokens <- tokens_select(train.tokens,
                              stopwords::stopwords("tr", source = "stopwords-iso"),
                              selection = "remove")
train.tokens[[123]]
```

```
##  [1] "ettiğin"                   "lafa"
##  [3] "kendinin"                  "inandığına"
##  [5] "inanmıyorum"               "atlantik"
##  [7] "ağzı"                      "abd"
##  [9] "dış"                       "ilişkiler"
## [11] "görevlisi"                 "misin"
## [13] "mübarek"                   "https://t.co/6e8jjc6syh"
```

```
# Perform stemming on the tokens.
```

```
train.tokens <- tokens_wordstem(train.tokens, language = "turkish")
train.tokens[[123]]
```

```
# Create our first bag-of-words model.
train.tokens.dfm <- dfm(train.tokens, tolower = FALSE)
dim(train.tokens.dfm)
```

```
## [1]  2100 15585
```

```
#Choosing only words with frequency >= 3
train.tokens.dfm_f <- dfm_trim(train.tokens.dfm, min_termfreq = 3)
dim(train.tokens.dfm_f)
```

```
## [1] 2100 1834
```

```
# weight by tf-idf
train.tokens.tfidf <- dfm_tfidf(train.tokens.dfm_f)
```

```
# Transform to a matrix and inspect.
train.tokens.matrix <- as.matrix(train.tokens.tfidf)
dim(train.tokens.matrix)
```

```
## [1] 2100 1834
```

```
#feature engineering by adding number of tokens in the tweets
ntoken_train <- ntoken(train.tokens)
train.tokens.matrix <- cbind(train.tokens.matrix, ntoken =  ntoken_train)
dim(train.tokens.matrix)
```

```
## [1] 2100 1835
```

```
# Setup a the feature data frame with labels.
train.tokens.df <- cbind(label = labels_train, data.frame(train.tokens.matrix))
train.tokens.df$label <- as.factor(train.tokens.df$label)
# Often, tokenization requires some additional pre-processing
# Cleanup column names.
names(train.tokens.df) <- make.names(names(train.tokens.df))
```

```
# Tokenize tweet texts
test.tokens <- tokens(test$text, what = "word",
                       remove_numbers = TRUE, remove_punct = TRUE,
                       remove_symbols = TRUE, split_hyphens =TRUE)
# Lower case the tokens.
test.tokens <- tokens_tolower(test.tokens)
# Use a specific stopword list from stopwords-iso dictionary for Turkish text
test.tokens <- tokens_select(test.tokens,
                              stopwords::stopwords("tr", source = "stopwords-iso"),
                              selection = "remove")
# Create our first bag-of-words model.
test.tokens.dfm <- dfm(test.tokens, tolower = FALSE)
#Feature Reduction
#Choosing only words with frequency >= 2
test.tokens.dfm_f <- dfm_trim(test.tokens.dfm, min_termfreq = 2)
#weigth by tf-idf
test.tokens.tfidf <- dfm_tfidf(test.tokens.dfm_f)
# Transform to a matrix and inspect.
test.tokens.matrix <- as.matrix(test.tokens.tfidf)
#feature engineering by adding number of tokens in the tweets
ntoken_test <- ntoken(test.tokens)
```

```
test.tokens.matrix <- cbind(test.tokens.matrix, ntoken = ntoken_test)
# Setup a the feature data frame with labels.
test.tokens.df <- cbind(label = labels_test, data.frame(test.tokens.matrix))
test.tokens.df$label <- as.factor(test.tokens.df$label)
# Often, tokenization requires some additional pre-processing
# Cleanup column names.
names(test.tokens.df) <- make.names(names(test.tokens.df))
```

```
dim(test.tokens.dfm)
```

```
## [1]  900 7879
```

```
dim(test.tokens.dfm_f)
```

```
## [1]  900 1594
```

```
dim(test.tokens.matrix)
```

```
## [1]  900 1595
```

4 Building Classification Models

4.1 Build Models

- which algorithms will you use and why? naive bayes, knn, decision trees, rules, log regression, multi regression, lasso, ridge, neural net, svm, clustering
- is the algorithm compatible with the features you have in the data set?

I will build Naive Bayes, kNN clustering, Random Forest and Support Vector Machines models to predict the classes based on text data. As these are most commonly applied classification algorithms, I decided to compare their utility and put all of them into a stacked model at the end.

```
#load neccesary library
library(e1071)
#since naive bayes classifier need categorical variables
#we are converting the numerical features to categorical using a custom function
makeCategoric <- function(x) {
 x <- ifelse(x == 0, "out", "in")
 }
trainLab.nominal <- ifelse(labels_train == 1, "1", "0")
testLab.nominal <- ifelse(labels_test == 1, "1", "0")
convert_counts <- function(x) {
  x <- ifelse(x > 0, "Yes", "No")
}
train_NB <- apply(train.tokens.matrix, MARGIN = 2,convert_counts)
test_NB <- apply(test.tokens.matrix, MARGIN = 2,convert_counts)

#training the model
NBclassifier <- naiveBayes(train_NB, trainLab.nominal)

#predicting the documents type
pred_NB <- predict(NBclassifier, test_NB)
```

```
#Different from Naive Bayes model, Random Forest Model needs that both train and test datasets have the
# We have two options:
# - one is cleaning first, splitting second.
# - the other is splitting first, cleanind second.(I will use the second option in this model)
# #At first, we can do cleaning stuff first, then split the dataset
# data.tokens <- tokens(data$text, what = "word",
#                        remove_numbers = TRUE, remove_punct = TRUE,
#                        remove_symbols = TRUE, split_hyphens    =TRUE)
# #Lower case the tokens.
# data.tokens <- tokens_tolower(data.tokens)
# #Use a specific stopword list from stopwords-iso dictionary for Turkish text
# data.tokens <- tokens_select(data.tokens,
#                               stopwords::stopwords("tr", source = "stopwords-iso"),
#                               selection = "remove")
# #Create our first bag-of-words model.
# data.tokens.dfm <- dfm(data.tokens, tolower = FALSE)
# #Feature Reduction
# #Choosing only words with frequency >= 2
# data.tokens.dfm_f <- dfm_trim(data.tokens.dfm, min_termfreq = 2)
# #weigth by tf-idf
# data.tokens.tfidf <- dfm_tfidf(data.tokens.dfm_f)
# #Transform to a matrix and inspect.
# data.tokens.matrix <- as.matrix(data.tokens.dfm_f)
# #feature engineering by adding number of tokens in the tweets
# ntoken_data <- ntoken(data.tokens)
# data.tokens.matrix <- cbind(data.tokens.matrix, ntoken_data)
# #Setup a the feature data frame with labels.
# data.tokens.df <- cbind(label = data$label, data.frame(data.tokens.matrix))
# data.tokens.df$label <- as.factor(data.tokens.df$label)
# train_RF <- data.tokens.df[trainRows,]
# test_RF <- data.tokens.df[-trainRows,]
# labels_train <- data.tokens.df[trainRows,1]
# labels_test <- data.tokens.df[-trainRows,1]

#In this model, I will split the dataset first, then do cleaning stuff.
#test_dfm_kNN <- dfm_select(test.tokens.dfm_f, pattern = colnamestrain, selection = "keep")
#I will use dfm_match function here to make dimensions equal
test_dfm_RF <- dfm_match(test.tokens.dfm_f, colnames(train.tokens.dfm_f))
test_matrix_RF <- as.matrix(test_dfm_RF)
ntoken_test <- ntoken(test.tokens)
test_matrix_RF <- cbind(test_matrix_RF, ntoken = ntoken_test)
test_RF <- cbind(label = labels_test, data.frame(test_matrix_RF))
test_RF$label <- as.factor(test_RF$label)
train_RF <- train.tokens.df

#build Random Forest classifier
RFclassifier <- train(label ~ ., data = train_RF, method = "rpart",
                      trControl = cv.cntrl, tuneLength = 2)

#predicting the documents type
pred_RF <- predict(RFclassifier, test_RF)
```

```
#we should use the same train and test dataset we prepared for Random Forest
#for the same reason. SVM only accepts datasets with the same dimensions
set.seed(1071)
#build simple models
SVM_lin <- svm(label~., data=train_RF, cost=10, kernel="linear")
```

```
SVM_rad <- svm(label~., data=train_RF, cost=10, kernel="radial")
#create cost values for hyperparameter tuning and find best model
#costvalues <- 10^(-2:1)
# SVM_lin_tuned <- tune(svm, label~., data=train_RF,
#                     kernel="linear",
#                    ranges=list(cost=costvalues))
#SVM_rad_tuned <- tune(svm, label~., data=train_RF_1,
#                     ranges=list(cost=costvalues),
#                     kernel="radial")

pred_SVM_lin <- predict(SVM_lin,newdata=test_RF)
pred_SVM_rad <- predict(SVM_rad,newdata=test_RF)
#pred_SVM_lintuned <- predict(SVM_lin_tuned$best.model,newdata=test_RF)
```

```
set.seed(1836)
#we can use the train function from caret package but I class library here.
#kNNclassifier <- train(label ~ ., data = train_RF_2, method = "knn")
#predicting the documents type
#pred_kNN <- predict(kNNclassifier, test_RF_2)
#prediction using knn model from class library
pred_kNN = class::knn(train_RF[,-1],test_RF[,-1], labels_train,k=2)
```

4.2 Ensemble Model

- how would you build a stacked ensemble model? is it a better model? can you use boosting or bagging? or build a stacked learner?

```
#### 3. Generate stacking model ####
stackedLearner <- function(pred_NB, pred_kNN, pred_RF, pred_SVM_lin){
#convert the labels into the numeric format
nb <- as.numeric(pred_NB)
kNN <- as.numeric(pred_kNN)
rf <- as.numeric(pred_RF)
svm <- as.numeric(pred_SVM_lin)
#merged them into one data.frame
merged <- data.frame(nb, kNN, rf, svm)
convert <- function(x) {
    x <- ifelse(x == 2, "1", "0")
}
merged <- apply(merged,2,convert)
merged <- apply(merged, 2, as.numeric)
#get rowsums for voting
rowSums <- rowSums(merged)
merged <- data.frame(merged, rowSums)
merged$stacked <- 0
#write a for loop to get stacked model's labels
for(i in 1:900){
  if(merged$rowSums[i]>=3){
    merged$stacked[i]=1
  }
  else {
    merged$stacked[i]=0
  }
}
pred_Stacked <- merged$stacked
#return predictions from stacked model
return(pred_Stacked)
}
```

```
#apply the prediction function and look at the disctribution
pred_Stacked <- stackedLearner(pred_NB, pred_kNN, pred_RF, pred_SVM_lin)
table(stackedLearner(pred_NB, pred_kNN, pred_RF, pred_SVM_lin))
```

```
##
##   0   1
## 245 655
```

5 Performance Evaluation

how do you compare and evaluate the performance of the algorithms? R-Squared? MAD, MSE, RMSE? AIC? AUC? why are they that way?

I will evaluate precision, senstivity, specificity and F-1 scores. Most of the metrics are provided by the confusionMatrix function however I apply a function generating the precision, recall and F-1 scores as an output to compare different models.

- The precision (the positive predictive value): the proportion of positive examples that are truly positive
- The sensitivity of the model (the true positive rate- recall): a measure of how complete the results are: the number of true positives over the total number of positives.
- The specificity of the model (the true negative rate)
- A measure of model performance that combines precision and recall into a single number is known as the F-measure (also sometimes called the F 1 score or the F-score). The F-measure combines precision and recall using the harmonic mean, a type of average that is used for rates of change.

```
# Custom function for calculation of precision, recall and F-1 metrics:
metrics <- function(confMatrix) {
  TP <- confMatrix$table[1][1]
  FP <- confMatrix$table[2][1]
  FN <- confMatrix$table[3][1]
  TN <- confMatrix$table[4][1]

  model.precision <- TP / (TP + FP)
  model.recall <- TP / (TP + FN)
  f1 <- (2*model.precision*model.recall)/(model.precision + model.recall)
  performance.metrics <- c(model.precision, model.recall, f1)
  return(performance.metrics)
}
```

```
cm_NB <- confusionMatrix(as.factor(pred_NB),
                reference =  as.factor(testLab.nominal),
                positive = "0")
cm_NB
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 136  72
##          1  49 643
##
```

```
##                Accuracy : 0.8656
##                  95% CI : (0.8415, 0.8872)
##     No Information Rate : 0.7944
##     P-Value [Acc > NIR] : 1.936e-08
##
##                   Kappa : 0.6065
##
##  Mcnemar's Test P-Value : 0.0455
##
##             Sensitivity : 0.7351
##             Specificity : 0.8993
##          Pos Pred Value : 0.6538
##          Neg Pred Value : 0.9292
##              Prevalence : 0.2056
##          Detection Rate : 0.1511
##    Detection Prevalence : 0.2311
##       Balanced Accuracy : 0.8172
##
##        'Positive' Class : 0
##
```

```
#compare different models
cm_RF <- confusionMatrix(as.factor(pred_RF),
                          reference =  as.factor(labels_test), positive = "0")
cm_RF
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  11   0
##          1 174 715
##
##                Accuracy : 0.8067
##                  95% CI : (0.7793, 0.832)
##     No Information Rate : 0.7944
##     P-Value [Acc > NIR] : 0.1938
##
##                   Kappa : 0.0913
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.05946
##             Specificity : 1.00000
##          Pos Pred Value : 1.00000
##          Neg Pred Value : 0.80427
##              Prevalence : 0.20556
##          Detection Rate : 0.01222
##    Detection Prevalence : 0.01222
##       Balanced Accuracy : 0.52973
##
##        'Positive' Class : 0
##
```

```r
cm_kNN <- confusionMatrix(as.factor(pred_kNN),
                reference =  as.factor(labels_test), positive = "0")
cm_kNN
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction   0   1
##          0  97 310
##          1  88 405
##                                           
##                Accuracy : 0.5578          
##                  95% CI : (0.5246, 0.5905)
##     No Information Rate : 0.7944          
##     P-Value [Acc > NIR] : 1               
##                                           
##                   Kappa : 0.0628          
##                                           
##  Mcnemar's Test P-Value : <2e-16          
##                                           
##             Sensitivity : 0.5243          
##             Specificity : 0.5664          
##          Pos Pred Value : 0.2383          
##          Neg Pred Value : 0.8215          
##              Prevalence : 0.2056          
##          Detection Rate : 0.1078          
##    Detection Prevalence : 0.4522          
##       Balanced Accuracy : 0.5454          
##                                           
##        'Positive' Class : 0               
## 
```

```r
cm_SVM_lin <- confusionMatrix(as.factor(pred_SVM_lin),
                              reference =   as.factor(labels_test),
                              positive = "0")
cm_SVM_rad <- confusionMatrix(as.factor(pred_SVM_rad),
                              reference =   as.factor(labels_test),
                              positive = "0")

# cm_SVM_lintuned <- confusionMatrix(as.factor(pred_SVM_lintuned),
#                                reference =   as.factor(labels_test),
#                                positive = "0")

cm_SVM_lin
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction   0   1
##          0  91 160
##          1  94 555
##                                          
##                Accuracy : 0.7178         
##                  95% CI : (0.6871, 0.747)
##     No Information Rate : 0.7944
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2368
##
##  Mcnemar's Test P-Value : 4.533e-05
##
##             Sensitivity : 0.4919
##             Specificity : 0.7762
##          Pos Pred Value : 0.3625
##          Neg Pred Value : 0.8552
##              Prevalence : 0.2056
##          Detection Rate : 0.1011
##    Detection Prevalence : 0.2789
##       Balanced Accuracy : 0.6341
##
##        'Positive' Class : 0
##
```

```
cm_SVM_rad
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  70  28
##          1 115 687
##
##                Accuracy : 0.8411
##                  95% CI : (0.8156, 0.8644)
##     No Information Rate : 0.7944
##     P-Value [Acc > NIR] : 0.0002149
##
##                   Kappa : 0.4108
##
##  Mcnemar's Test P-Value : 6.4e-13
##
##             Sensitivity : 0.37838
##             Specificity : 0.96084
##          Pos Pred Value : 0.71429
##          Neg Pred Value : 0.85661
##              Prevalence : 0.20556
##          Detection Rate : 0.07778
##    Detection Prevalence : 0.10889
##       Balanced Accuracy : 0.66961
##
##        'Positive' Class : 0
##
```

```
#cm_SVM_lintuned
```

```
cm_stacked <- confusionMatrix(as.factor(pred_Stacked), as.factor(labels_test))
cm_stacked
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction   0   1
##          0 113 132
##          1  72 583
##                                           
##                Accuracy : 0.7733          
##                  95% CI : (0.7446, 0.8003)
##     No Information Rate : 0.7944          
##     P-Value [Acc > NIR] : 0.9448          
##                                           
##                   Kappa : 0.3805          
##                                           
##  Mcnemar's Test P-Value : 3.615e-05       
##                                           
##             Sensitivity : 0.6108          
##             Specificity : 0.8154          
##          Pos Pred Value : 0.4612          
##          Neg Pred Value : 0.8901          
##              Prevalence : 0.2056          
##          Detection Rate : 0.1256          
##    Detection Prevalence : 0.2722          
##       Balanced Accuracy : 0.7131          
##                                           
##        'Positive' Class : 0               
## 
```

```
nb <-  metrics(cm_NB)
rf <- metrics(cm_RF)
knn <- metrics(cm_kNN)
svm_l <- metrics(cm_SVM_lin)
svm_r <- metrics(cm_SVM_rad)
#svm_lt <- metrics(cm_SVM_lintuned)
stacked <- metrics(cm_stacked)
colname <- c("model", "precision", "recall", "fscore")
rowname <- c("nb", "rf", "knn", "svm_r","stacked")
evaltable <- rbind(nb, rf, knn, svm_l, svm_r, stacked)
evaltable <- as.data.frame(cbind(rowname, evaltable))
names(evaltable) <- colname
evaltable
```

	model	precision	recall	fscore
nb	nb	0.735135135135135	0.653846153846154	0.692111959287532
rf	rf	0.0594594594594595	1	0.112244897959184
knn	knn	0.524324324324324	0.238329238329238	0.327702702702703
svm_l	svm_r	0.491891891891892	0.362549800796813	0.41743119266055
svm_r	stacked	0.378378378378378	0.714285714285714	0.49469964664311
stacked	nb	0.610810810810811	0.461224489795918	0.525581395348837

6 Conclusionary Remarks

- how will you communicate the results of your algorithms?

I am searching for irrelevant tweets in my dataset. Therefore, I need to filter positive cases(being 0) more accurately but I do not want to sacrifice relevant tweets during that process. Comparing all aghorithms and the ensemble learner, I argue that Naive Bayesian classifer is the most useful and easy to implement model compared to other. All other algorithms requires that the dimensions of both training and test dataset should be equal. This is not a so complex process and but it reduces the performance of models.