Group final project - BAN400 fall 2020

Candidate: 40 & ..

15/12/2020

```r
# loading the required packages:
require(tidyverse)
require(parsnip)
require(yardstick)
require(rsample)
require(doParallel)
require(recipes)
require(ggridges)
require(ggplot2)
require(patchwork)
require(xgboost)
require(tidymodels)
require(FactoMineR)
require(tibble)
require(tm)
require(tidytext)
require(sentimentr)
require(SentimentAnalysis)
require(lexicon)
require(tokenizers)
require(wordcloud)
require(docstring)

### IMPORTANT! ###
# Set the number of cores you will use for parallel calculation.
# Do not exceed the recommended amount.

CORES <- 8
cl <- parallel::makeCluster(CORES)
```

Contents

Project suggestion Group 30

Initial proposal:

“Textual data analysis combined with regression analysis using data on fake/true news”

Method:

- We create regressors based on the words present in the fake and real news. The regressors are based on both the formatting of the article, such as: *Number of words in the body and title of the article; Number of punctuation characters in the body and title; Number of exclamation marks in the body and title; Number of characters in upper case in the body and title.* And on the words present within the article, such as: *Words expressing positive or negative sentiment, words relating to relevant American politicians, words that relate to media itself.*
- Then use a predictive model to see if the news are real or fake based on these factors. We run multiple models and choose the one with the best predictive score against the test dataset.
- Finally, we can create a “shiny ap” which allows up to paste news articles in, the ap will then preprocess the article and give us a score of “fake probability”.

Analysis:

- *Tokenizing and prepossessing before doing any textual analysis.* We need to prepossess the data, which mean shaping it in order for the different models to read them. Before that, we store the peculiarities of the text (such as the amount of punctiation characters present, for example).
- *Visualization of the data.* Since we are working with a large corpus (body of text), it is important to employ some techniques in order to better understand what the text we are using is talking about. We use frequency tables, wordclouds and KWIC (Keyword in context) tables to define what words are likely to carry information.
- *Sentiment analysis.* It is used to gather data on how negative /positive the fake news is compared with true news. Maybe there is a specific sentiment present which would add value to our analysis.
- *Regression*, where 1 is fake news and 0 is true news. First we use a linear regression to understand the relationship between our regressors and our prediction. Then we run multiple machine learning algorithms, and we keep whichever performs best.

Sources

- For this we will use a data set from Kaggle: https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset (Links to an external site.) with data from 2016 to 2017.

- All graphs will be visualized using ggplot package.

Sentiment dictionary:

- Loughran, T. and McDonald, B. (2016). Textual analysis in accounting and finance: A survey. Journal of Accounting Research 54(4), 1187-1230. doi: 10.2139/ssrn.2504147

Workflow:

1. Getting the data to run in R and tidyverse

```r
# Loading in the data, binding them together----

df_fake <- readRDS("Fake.rds") %>%
  mutate(Fake = 1)

df_true <- readRDS("True.rds") %>%
  mutate(Fake = 0)
```

2. Merging true and fake news datasets, adding a new dummy column if news is true or fake.

```r
DF <- rbind(df_fake, df_true)

rm(df_true, df_fake)
```

3. Prepossesing and getting the data ready for modelling.

3.1 Extracting counts of the formatting choices

Prepossesing the data means removing most of the formatting. However, we will use some formatting choices as regressors, so before shaping the text we need to extract the information we need.

```r
# First we'd like to extract a few variables
# which we might use on our predictive models

DF <- DF %>%  #here we are counting space as a proxy for word count.
  mutate(w_count.txt = str_count(paste(DF$text), " "),
         punct_count.txt = str_count(paste(DF$text), "[[:punct:]]"),
         upper_count.txt = str_count(paste(DF$text), "[[:upper:]]"),
         exclamation_count.txt = str_count(paste(DF$text), "!"),
         w_count.title = str_count(paste(DF$title), " "),
         punct_count.title = str_count(paste(DF$title), "[[:punct:]]"),
         upper_count.title = str_count(paste(DF$title), "[[:upper:]]"),
         exclamation_count.title = str_count(paste(DF$title), "!"))
```

3.2 Prepossessing the data

Now we can preposses the data. In the case of our corpus, we:

- Removed datapoints with no title, and bodies with less than 30 words.
- Removed uppercase letters.
- Removed any character that isn't alphabetical
- Removed any words that were shorter than 2 letters and larger than 21 letters
- Removed stopwords (words that are frequent but carry no meaning).

```r
# Now we do some prepossessing of the data

DF <-  DF %>%
  filter( w_count.title > 0 ) %>%
  filter( w_count.txt > 30 ) %>%
  mutate(text = text %>% tolower() %>%
             gsub("[^[:alpha:]]", " ", .) %>%
#this removes any character that's not alphabetic.
#Those usually not carry any meaning. We may want to analyze them individually.
             gsub('\\b\\w{1,2}\\s|\\b\\w{21,}\\s','', .) %>%
#removing words that are too small (from 1 or 2 letters) and huge words,
  #(21+ letters)
             removeWords(., c(stopwords(kind = "en"))),
# removing stopwords, which are words that are most common in the English
# language, yet carry no meaning. These are found inside the stopwords
# dictionary in the tm package.
         title = title %>% tolower() %>%
             gsub("[^[:alpha:]]", " ", .) %>%
             gsub('\\b\\w{1,2}\\s|\\b\\w{21,}\\s','', .) %>%
             removeWords(., c(stopwords(kind = "en"))))
```

3.3 Tokenizing the data and creating bigrams

Here we will split the data word by word. We will do the analysis with unigrams (which is every word separated individually), and bigrams (which are words separated in twos). The individual words will be used for the sentiment analysis, while bigrams will give us better keywords.

```r
# Tokenizing

# Tokenizing with bigrams. What this does to us, is that as bigrams,
# words carry a different meaning as the paired words may carry a different
# meaning.

DF <-  DF %>%
  mutate(bigram_text = tokenize_ngrams(text , n = 2, ngram_delim = "_"),
         bigram_title = tokenize_ngrams(title , n = 2, ngram_delim = "_"),
         token_text = tokenize_ngrams(text , n = 1, ngram_delim = ""),
         token_title = tokenize_ngrams(title , n = 1, ngram_delim = ""))
```

3.4 Adding the sentiment score

The code will run through every single word in our text corpus and rate them with a sentiment ranging from -1 (bad) to 1 (good). Then it will return the average score for each article.

NOTE: Adding the sentiment score takes a long time, *over 2 hours with 8 cores*, the sentiment.rds file should have been included with this code, and it contains the data we get after we run the code. If you would prefer not to wait for the code to run, *skip the next chunk*.

```r
sentiment <- readRDS(file = "sentiment.rds")

# Adding the sentiment score to our DF

DF$sentiment <-sentiment

# making sure there are no NA's left in the sentiment score.

DF <- DF %>%
  na.omit(DF$sentiment)
```

3.5 Adding more predictors based on our visualization

We added this step after performing the visualization. Here we create a score based on the keyword topics we decided after looking at the data. We decided to create a politician per word score, and a media per word score. We decided that bigrams would carry more meaning in our analysis, and they were picked based on what we saw within the training data and what we believe to be relevant. This does mean that there is a higher potential for overfitting these predictors to the training data.

The bigrams we chose for politicians were:

- *barack_obama, bernie_sanders, donald_trump, hillary_clinton and joe_biden*

The bigrams we chose for media were:

- *twitter_com, pic_twitter, getty_images, fox_news, social_media, fake_news, york_times, washington_post, mainstream_media, via_youtube, via_getty, via_video, facebook_page, youtube_com and abc_news*

```
#Searching for certain keywords

#Politicians relevant at the time
politicians <- "(barack_obama)|(bernie_sanders)|(donald_trump)|
                (hillary_clinton)|(joe_biden)"

# Creating a politician per word metric, using the bigrams
DF$politician.pw <-
  str_count(string = paste(DF$bigram_text),
            pattern = politicians)/DF$w_count.txt

#Media related bigrams
media <- "(twitter_com)|(pic_twitter)|(getty_images)|(fox_news)|(social_media)|
          (fake_news)|(york_times)|(washington_post)|(mainstream_media)|
          (via_youtube)|(via_getty)|(via_video)|(facebook_page)|(youtube_com)|
          (abc_news)"

# Creating a politician per word metric, using the bigrams
DF$media.pw <-
  str_count(string = paste(DF$bigram_text),
            pattern = media)/DF$w_count.txt
```

4. Separate the data into training and test data.

```r
# Setting a seed so the experiment is replicable

set.seed(10)

# Creating test and train data:

sample <- initial_split(DF,
                        strata = Fake, # variable used for stratified sampling
                        prop = 0.75)# We are splitting 75% training and 25% test.

# Extracting the training data:

train<-training(sample)

# Extracting the test data:

test<-testing(sample)
```

Here we split our data into training and test. First, we set a seed so the test is replicable. Then the split is made with 75% of our data in the training set, and 25% on the test set. The data is stratified with the Fake dummy variable in mind. It is important we go through this step before visualizing the data. Because we create some variables through visualization, and we don't want to introduce bias into our analysis.

5. Visualizing the data

Here we have the steps we took to see how our data actually looked like. Since this data is filled with so much text, we used a Key words in context table to see specific words in the context of where they appear. We also can visualize the frequency of the words in a wordcloud, to better understand what is being most spoke on in our dataset.

```r
#splitting the data back into true and false to see the difference between them.
f.train <- train  %>% filter(train$Fake == 1)
t.train <- train  %>% filter(train$Fake == 0)

# extracting all the words from our training data to create the topic modeling
f.all_text <- paste(c(f.train$text), collapse = " ")
f.all_title <- paste(c(f.train$title), collapse = " ")
t.all_text <- paste(c(t.train$text), collapse = " ")
t.all_title <- paste(c(t.train$title), collapse = " ")

# splitting the words and tokenizing them.
f.text_tokens <- scan(text = f.all_text, what = "character", quote = "")
f.title_tokens <- scan(text = f.all_title, what = "character", quote = "")

f.text_bigrams <- paste(unlist(f.train$bigram_text), collapse = " ") %>%
  strsplit(.," ") %>% unlist
f.title_bigrams <- paste(unlist(f.train$bigram_title), collapse = " ") %>%
  strsplit(.," ") %>%  unlist

t.text_tokens <- scan(text = t.all_text,
                      what = "character", quote = "")
t.title_tokens <- scan(text = t.all_title,
                       what = "character", quote = "")

t.text_bigrams <- paste(unlist(t.train$bigram_text), collapse = " ") %>%
  strsplit(.," ") %>% unlist
t.title_bigrams <- paste(unlist(t.train$bigram_title), collapse = " ") %>%
  strsplit(.," ") %>%  unlist
```

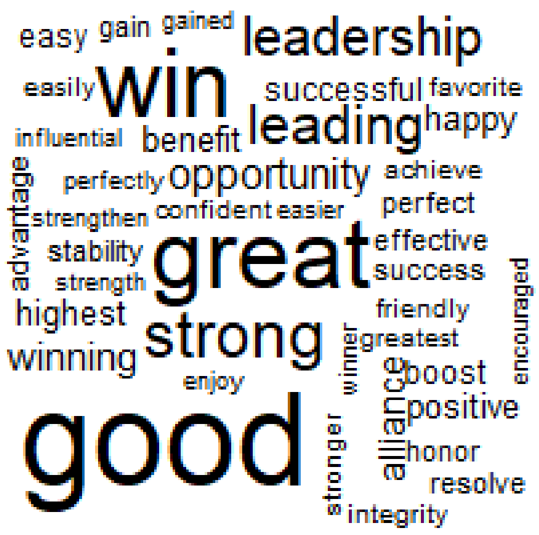
5.1 Visualizing the sentiment analysis

For the sentiment analysis, we decided to go with the sentiment dictionary: Loughran-McDonald Polarity Table. We chose that dictionary after looking at the words present in our data, and checking them in the KWIC matrix to see them in context. This dictionary seemed to be the more neutral one which made most sense to us. We found this dictionary in the lexicon package, and its description goes as follows: "*A data.table dataset containing an filtered version of Loughran & McDonald's (2016) positive/negative financial word list as sentiment lookup values.*"

```r
# Creating a dataframe with all the words
all_text <- paste(c(train$text), collapse = " ")

#tokenizing all words
text_tokens <- scan(text = all_text,
                     what = "character",
                     quote = "")

text_bigrams <- paste(unlist(train$bigram_text), collapse = " ") %>%
  strsplit(.," ") %>% unlist

#Making a frequency table of all words

freq_text <- table(text_tokens)%>%
  sort(decreasing = T) %>%
  as.data.frame()

# choosing the sentiment dictionary
dictionary <- lexicon::hash_sentiment_loughran_mcdonald
dictionary.n <- dictionary%>% filter(dictionary$y < 0)
dictionary.p <- dictionary%>% filter(dictionary$y > 0)

#seeing which positive words are present
positive_words <- freq_text %>%
  filter(text_tokens %in% dictionary.p$x) %>%
  arrange(desc(Freq)) %>% head(50)

# seeing which negative words are present
negative_words <- freq_text %>%
  filter(text_tokens %in% dictionary.n$x) %>%
  arrange(desc(Freq)) %>% head(50)
```

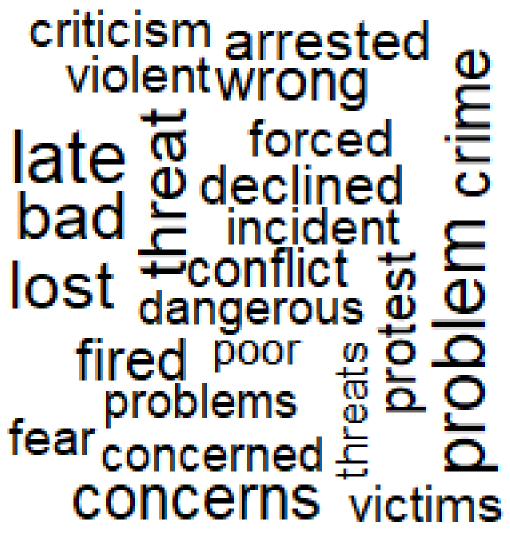
Positive words present in our data

```r
wordcloud(words = positive_words$text_tokens [1:50],
          freq = positive_words$Freq[1:50])
```



Negative words present in our data

```r
wordcloud(words = negative_words$text_tokens [1:50],
          freq = negative_words$Freq[1:50])
```



5.2 Making a KWIC (Key words in context) table

This helps us better visualize the text and understand the context of some words.

```r
# Getting a KWIC matrix
# Here we build the function "get.KWIC" in order to inspect keywords.
# The KWIC matrix helps us identify keywords and the words
# related to it in our corpus. It aids in visualizing the data.

### building the function

get.KWIC <- function(keyword,
                     n,
                     tokenized_text,
                     cores){
  #' Get Key Word In Context
  #'
  #' @description This function searches for the string pattern inputted
  #' as "keyword" in your corpus, and returns the n words before it and
  #' after it, so you can understand the context that the keyword chosen
  #' is in.
  #'
  #' @param keyword A string pattern in quotation marks
  #' @param n The number of words that are to be returned before and
  #' after the keyword.
  #' @param tokenized_text the tokenized corpus.
  #' @param cores This function uses parallel computing, so the number
  #' of cores to be used should be set here.
  #' @usage get.KWIC(keyword = "keyword",
  #'       n = 3,
  #'       tokenized_text = DF$text_tokens, 4)
  #' @return The return is a table with the 3 words before and after
  #' "keyword" appears in your text.
  #' @note This function takes a while to run.

  cores <- min(detectCores(), cores)
  cl <- makeCluster(cores)
  registerDoParallel(cl)

  # this helps us seek the keyword
  index.env <- grep(keyword, tokenized_text)


  # this creates the table for us as a tibble.
  KWIC <- tibble(left = parSapply(cl, index.env,
                             function(i) {ifelse( i-n >0,
                               paste(tokenized_text[i-n:1],
                                                 collapse = " "),NA)}),
               keyword = tokenized_text[index.env],
               right = parSapply(cl, index.env,
                              function(i) {paste(tokenized_text[i+1:n],
                                                  collapse = " ")}))

  stopCluster(cl)
  # return
  return(KWIC)

}

?get.KWIC
```

```r
### using the function

get.KWIC(keyword = "trump",
         n = 3,
         tokenized_text = text_tokens, CORES) -> KWIC.trump

get.KWIC(keyword = "donald_trump",
         n = 2,
         tokenized_text = text_bigrams, CORES) -> KWIC.trump_bigram



# we just picked these topics since they are polarizing,
# but they are only here as an example of the function at work
```

Example of keyword in context: trump on the tokenized data

```
require(kableExtra)


kable(KWIC.trump[4:10,],booktabs = T)
```

left	keyword	right
trump realdonaldtrump december	trump	tweet went welll
petty infantile gibberish	trump	lack decency won
pollitt korencarpenter december	trump	new year eve
know love donald	trump	realdonaldtrump december nothing
love donald trump	realdonaldtrump	december nothing new
december nothing new	trump	years trump directed
new trump years	trump	directed messages enemies

Example of keyword in context: donald_trump on the bigram data

```
kable(KWIC.trump_bigram[4:10,],booktabs = T)
```

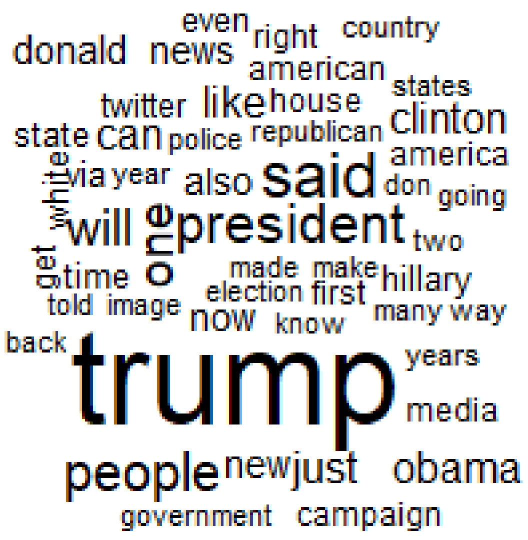
left	keyword	right
security_secretary secretary_donald	donald_trump	trump_administration administration_email
message_rebuke rebuke_donald	donald_trump	trump_without without_even
decision_derailed derailed_donald	donald_trump	trump_plan plan_bar
getty_images centerpiece_donald	donald_trump	trump_campaign campaign_now
break_wednesday wednesday_donald	donald_trump	trump_looked looked_cameras
mcconnell_knows knows_donald	donald_trump	trump_destroying destroying_gop
inequality_words words_donald	donald_trump	trump_paul paul_ryan

6 Visualizing word frequency

```r
#making frequency tables out of the fake news

f.freq_text_1 <- table(f.text_tokens)%>%
  sort(decreasing = T) %>%
  as.data.frame()

f.freq_text_2 <- table(f.text_bigrams)%>%
  sort(decreasing = T) %>%
  as.data.frame()

f.freq_title_1 <- table(f.title_tokens)%>%
  sort(decreasing = T) %>%
  as.data.frame()

f.freq_title_2 <- table(f.title_bigrams)%>%
  sort(decreasing = T) %>%
  as.data.frame()

#making frequency tables out of the true news

t.freq_text_1 <- table(t.text_tokens)%>%
  sort(decreasing = T) %>%
  as.data.frame()

t.freq_text_2 <- table(t.text_bigrams)%>%
  sort(decreasing = T) %>%
  as.data.frame()

t.freq_title_1 <- table(t.title_tokens)%>%
  sort(decreasing = T) %>%
  as.data.frame()

t.freq_title_2 <- table(t.title_bigrams)%>%
  sort(decreasing = T) %>%
  as.data.frame()
```

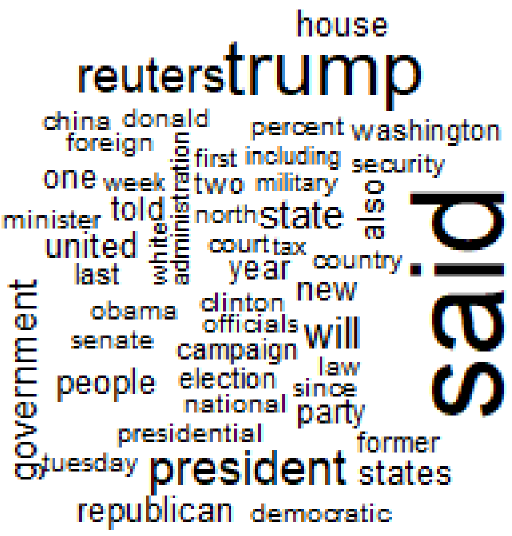
Fake News word frequency

```r
#Making wordclouds of our training data



wordcloud(words = f.freq_text_1$f.text_tokens [1:50],
          freq = f.freq_text_1$Freq[1:50])
```



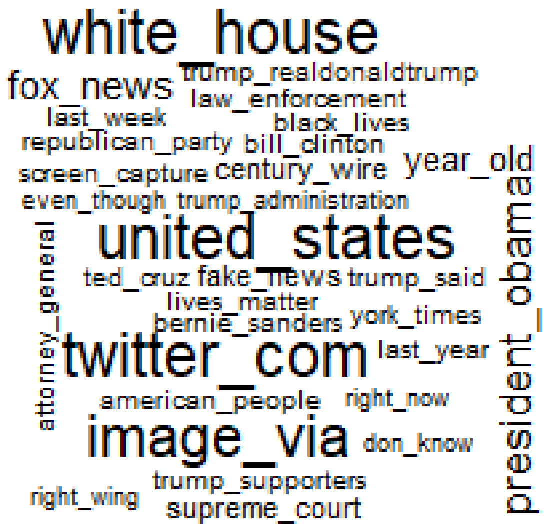
True News word frequency

```r
wordcloud(words = t.freq_text_1$t.text_tokens [1:50],
          freq = t.freq_text_1$Freq[1:50])
```



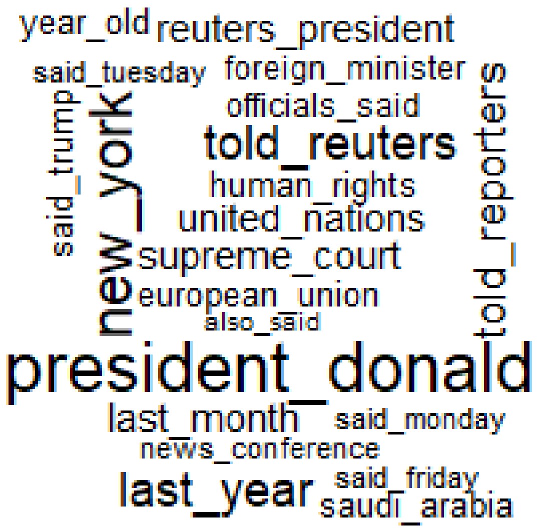
Fake News bigram frequency

```
wordcloud(words = f.freq_text_2$f.text_bigrams [1:50],
          freq = f.freq_text_2$Freq[1:50])
```



True News bigram frequency

```
wordcloud(words = t.freq_text_2$t.text_bigrams [1:50],
          freq = t.freq_text_2$Freq[1:50])
```



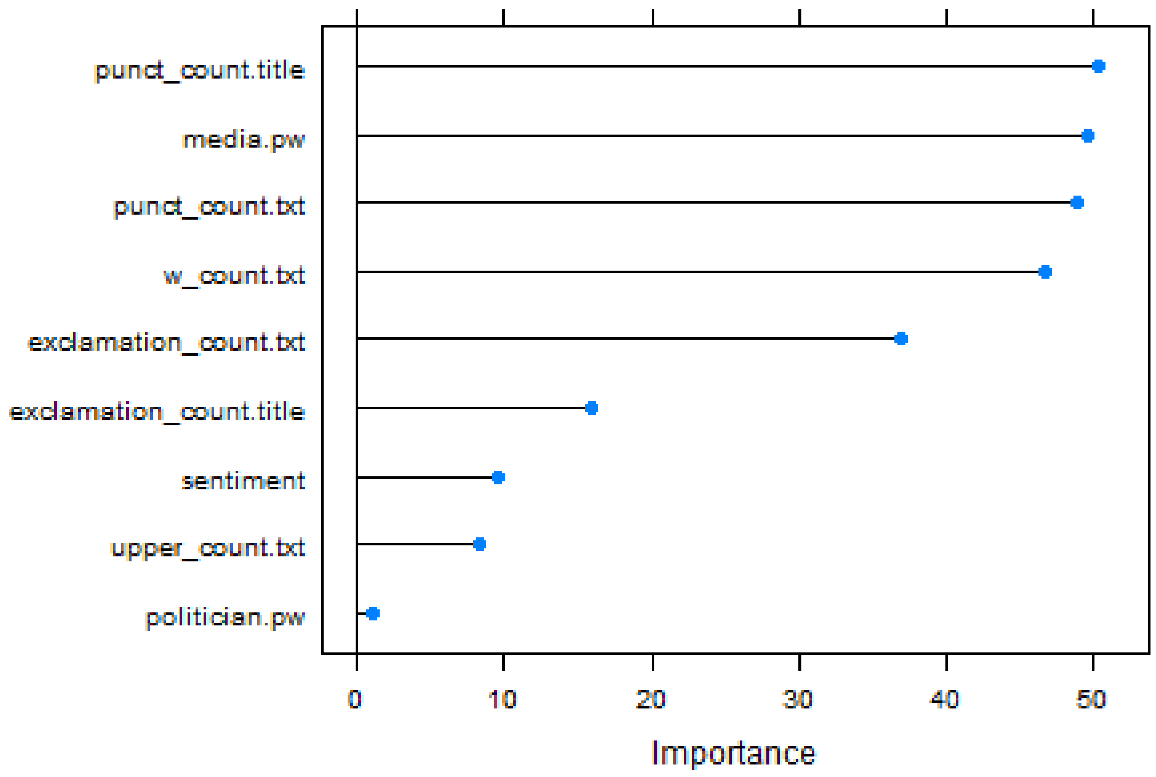
7. Training the machine learning model

```
require(tidyverse)
registerDoSEQ()

# Select all variable we are using in our models

model_train <- train %>%
  select(Fake,w_count.txt,punct_count.txt,exclamation_count.txt,
         #w_count.title,
         punct_count.title,exclamation_count.title,
         sentiment,politician.pw, #upper_count.title,
         media.pw, upper_count.txt  ) %>%
  mutate(Fake = as.factor(Fake))

model_test <- test %>%
  select(Fake,w_count.txt,punct_count.txt,exclamation_count.txt,
         #w_count.title,
         punct_count.title, exclamation_count.title,
         sentiment, politician.pw, #upper_count.title,
         media.pw, upper_count.txt ) %>%
  mutate(Fake = as.factor(Fake))
```

```
# Run logistic regression

require(caret)

# logistic reg

set.seed(55)

glm_mod <- train(
  form = Fake ~ .,
  data = model_train,
  trControl = trainControl(method = "cv", number = 5),
  method = "glm",
  family = "binomial"
)
```

Predictor importance plot

```
models <- c(list(glm_mod), readRDS("models.rds"))


# Variable importance plot

imp_lr <- varImp(models[[1]], scale = FALSE)

plot(imp_lr)
```



Here we can see how well our predictors performed in the logistic regression. It gives us a better understanding of how the data is shaped and what was most important on deciding if the news were fake or not. It seems like the formatting of the data gives off most information. While the sentiment analysis, despite being statistically relevant, does not give much information. We removed variables that had too high importance, despite it giving us a more accurate model. We did that since we felt that that would fit our model too much to the format that our data came in. We noticed this when testing a model with higher accuracy but variables with too high importance against articles we found online. Once we made any small changes on the dominating factors it would completely change the opinion of the model, and it made it feel less accurate. By removing those variables we manage to loosen the model and made it focus more on the text itself.

Testing the trained regression models

```
# Make function

summary_table <- function(models,testdata, var_of_interest){
  #' Create a Summary table
  #'
  #' @description This function will create a table with all the
  #' relevant information from testing the multiple models specified.
  #'
  #' @param models A trained model to be tested
  #' @param testdata The data the model should be tested against
  #' @param var_of_interest The regressed variable
  #' @usage sum_table <- summary_table(models, model_test,
  #' model_test$Fake)
  #' @return The return is a table with "Methods", "Accuracy", "Kappa",
  #' "Sensitivity" and "Specificity" scores.

  # making empty matrix to fill values

  sum_tab <- matrix(0,length(models),5)

  colnames(sum_tab) <- c("Methods", "Accuracy", "Kappa", "Sensitivity",
                         "Specificity")

  # Loop over all models

  for(i in 1:length(models)){

    # Predict and make a confusion matrix

    pred <- predict(models[[i]], testdata)

    cm <- confusionMatrix(pred, var_of_interest)

    # make a nice data table of the result

    sum_tab[i,1]<- models[[i]]$method
    sum_tab[i,2]<- cm$overall[1] %>% round(3)
    sum_tab[i,3]<- cm$overall[2] %>% round(3)
    sum_tab[i,4]<- cm$byClass[1] %>% round(3)
    sum_tab[i,5]<- cm$byClass[2] %>% round(3)

  }

  sum_tab

}


sum_table <- summary_table(models, model_test, model_test$Fake)

kableExtra::kable(sum_table, booktabs = T)
```

Methods	Accuracy	Kappa	Sensitivity	Specificity
glm	0.826	0.653	0.877	0.777
gbm	0.877	0.754	0.903	0.852
knn	0.745	0.489	0.74	0.749
xgbTree	0.856	0.713	0.905	0.81

Here we see that the GBM machine learning algorithm outperformed the other predictions. It also has a fairly high accuracy, albeit it is important to remember that this accuracy is very dependent on the fact that our test and training data comes from the same source.

8. Make a shiny app, which will work as an interface to plug in new data and see if the news stories is true or fake.

The codes used for the shiny app can be found in a separate folder. However, the app is uploaded into shinyapps.io and can be found on the following link:

https://cirk88.shinyapps.io/fake_app/

Happy exploring!

Conclusions

After finishing the app, we ran the code with different articles to see how the results would look like if we tested the data against articles that were not included in the original dataset. As it is mentioned in the app itself, this code does not check the validity of the content, but instead it analyzes the formatting of the text to see if it resembles the fake news included in our dataset. What this means, is that the same article can be called true or fake, depending on how we change small elements like punctuation or capital letters. Perhaps if we did a more in-depth research on keywords, and picked a more appropriate dictionary for the sentiment analysis our code would be able to more correctly guess the validity of true and fake news. As it stands, the app is a fun experiment, and it can be used to see if the formatting of an article resembles fake news or not.