Russia Troll Analysis

Estelle Pan

2024-02-28

install.packages("quanteda") install.packages("sentimentr") install.packages("quanteda.textmodels") install.packages("quanteda.textplots") install.packages("quanteda.textstats")

```
library(quanteda)
```

```
## Package version: 3.3.1
## Unicode version: 14.0
## ICU version: 71.1

## Parallel computing: 8 of 8 threads used.

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

```
library(ggplot2)
library(sentimentr)
library(quanteda.textplots)
```

```
## Warning: package 'quanteda.textplots' was built under R version 4.3.2
```

```
library(quanteda.textmodels)
library(quanteda.textstats)

# clean environment
rm(list=ls())

## Set working directory
setwd("/Users/minpan/Desktop/Text as Data")

## Read dataset
base::load("russian_trolls_sample.rdata")
```

Data source

The dataset, named "Russian Troll Tweets (russian_trolls_sample.rdata)," originates from the Internet Research Agency (IRA), an organization commonly referred to as a "troll factory." Allegedly funded by the Russian government, the IRA has been implicated in various online influence and disinformation campaigns.

This corpus comprises tweets disseminated by accounts linked to the IRA between 2014 and 2017. The dataset is structured into 10,000 rows, with each row representing a distinct tweet, and is organized into five columns:

The dataset includes 10,000 rows (each row is a tweet), and five columns: • tweet_id: A unique identifier for the tweet • author: The screen name of the troll account • text: The text of the tweet • account_category: The label of the account: right troll or a left troll. – who label it? • date: The date in which the tweet was posted

```
# Create a corpus
russian_trolls_corpus = corpus(russian_trolls_sample)
# Inspect the corpus
 # summary(russian_trolls_corpus)
```

Dataset exploration

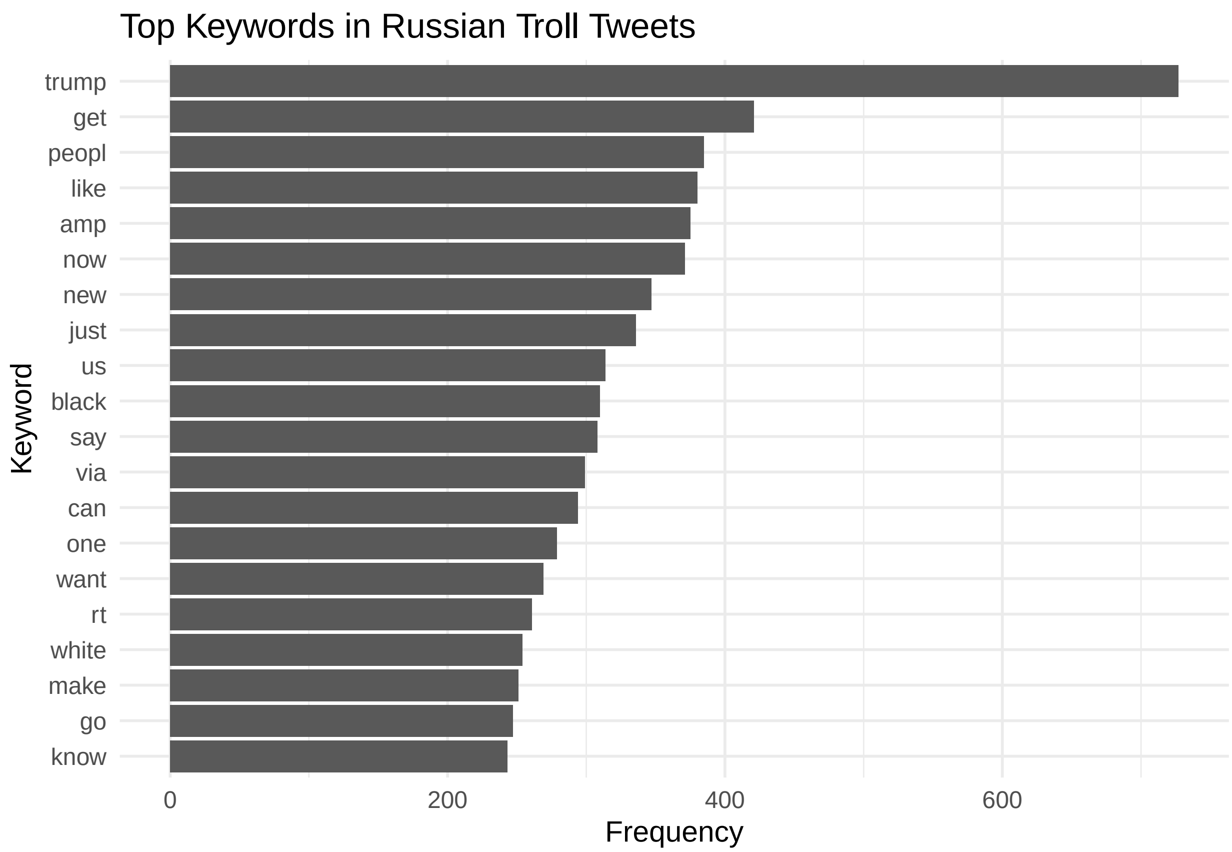
Word Frequency

```
# Create a DTM and Pre-Processing
russian_trolls_tokens = tokens(russian_trolls_corpus, remove_numbers = TRUE,
       remove_punct = TRUE, remove_url = TRUE,
       remove_symbols = TRUE) %>%
  tokens_tolower() %>%
  tokens_select(pattern = stopwords('en'), selection = 'remove') %>%
  tokens_wordstem(language = "en")

russian_trolls_dfm =  dfm(russian_trolls_tokens)

# Sum the columns of the DTM to get a total count for each word:
freqs = colSums(russian_trolls_dfm)
# Crate a vocabulary vector:
words = colnames(russian_trolls_dfm)
# Create a data frame that includes the words in the vocabulary and their frequencies:
wordlist = data.frame(words, freqs)
# Re-order the wordlist by decreasing frequency
wordlist = wordlist[order(wordlist[ ,"freqs"], decreasing = TRUE), ]
# What are the 10 most frequent words?
head(wordlist, 10)
```

```
##       words freqs
## trump trump   727
## get     get   421
## peopl peopl   385
## like   like   380
## amp     amp   375
## now     now   371
## new     new   347
## just   just   336
## us       us   314
## black black   310
```

Visualization 1: Bar chart

```r
# Calculate term frequency
term_frequency <- textstat_frequency(russian_trolls_dfm)

# Identify top keywords
top_keywords <- head(term_frequency, 20)

# Plot the top keywords
ggplot(top_keywords, aes(x = reorder(feature, frequency), y = frequency)) +
  geom_col() +
  coord_flip() +
  labs(x = "Keyword", y = "Frequency", title = "Top Keywords in Russian Troll Tweets") +
  theme_minimal()
```



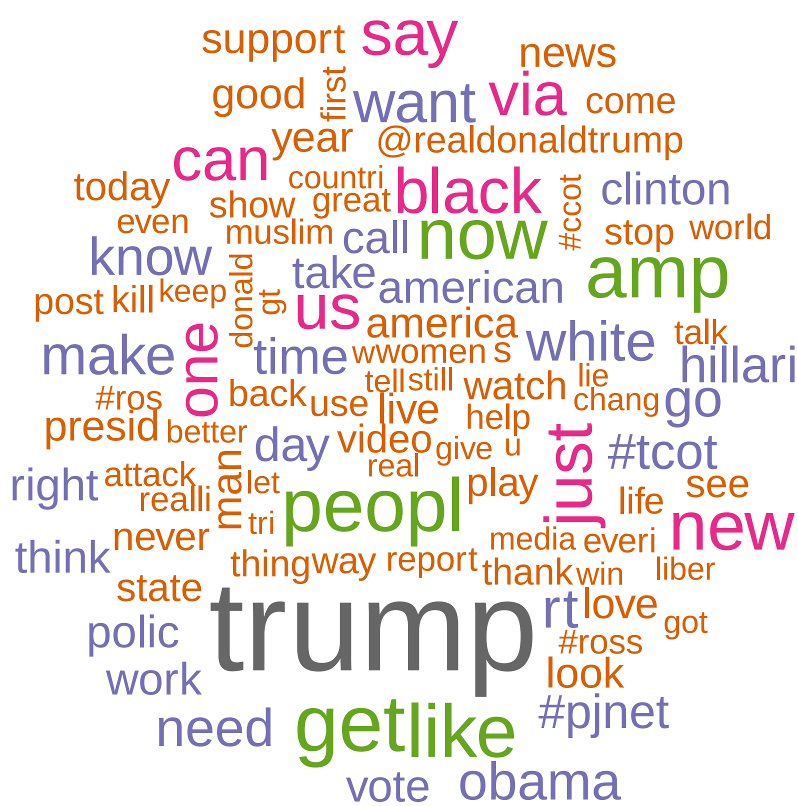
Visualization 2 - Wordcloud

install.packages("wordcloud")

```r
library(wordcloud)
```

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

```r
wordcloud(words = wordlist$words, freq = wordlist$freqs, max.words = 100 ,min.freq = 2,color = RColorBre
```



As we can see, "Trump" is the most frequently occurring word in the tweets associated with the Internet Research Agency (IRA). This suggests that discussions or content related to Donald Trump were prominently featured in the dataset.

Key Word Analysis

create dictionary on Ukranie

```
ukraine_dict <- dictionary(list(
    ukraine = c("trump", "zelensky", "ukraine", "russia", "nato", "eu", "putin", "invasion","security",
))

# Confirm the dictionary structure
print(ukraine_dict)
```

```
## Dictionary object with 1 key entry.
## - [ukraine]:
##   - trump, zelensky, ukraine, russia, nato, eu, putin, invasion, security, military, conflict, diplo
```

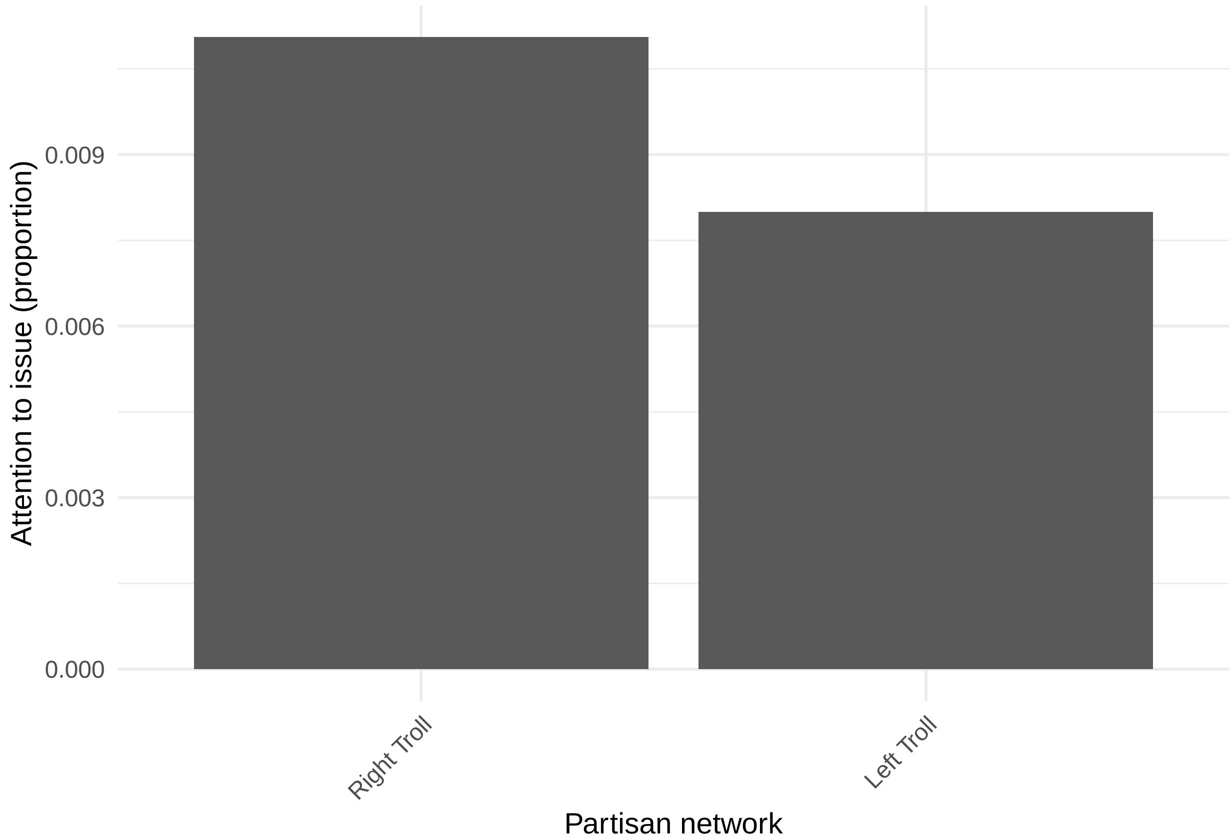
```
russian_trolls_tokens %>%
  tokens_lookup(dictionary = ukraine_dict) %>%
  dfm()
```

```
## Document-feature matrix of: 10,000 documents, 1 feature (92.14% sparse) and 4 docvars.
##        features
## docs    ukraine
##   text1       0
##   text2       0
##   text3       0
##   text4       0
##   text5       0
##   text6       0
## [ reached max_ndoc ... 9,994 more documents ]
```

```
right_troll_on_the_issue = tokens_subset(russian_trolls_tokens, subset= account_category == "RightTroll
  tokens_lookup(dictionary = ukraine_dict) %>%
  dfm() %>% sum()

right_troll_total_words = sum(russian_trolls_dfm[docvars(russian_trolls_dfm)$account_category == "Right
right_troll_on_the_issue = right_troll_on_the_issue/right_troll_total_words
```

```
left_troll_on_the_issue = tokens_subset(russian_trolls_tokens, subset=account_category == "LeftTroll") %
  tokens_lookup(dictionary = ukraine_dict) %>%
  dfm() %>% sum()

left_troll_total_words = sum(russian_trolls_dfm[docvars(russian_trolls_dfm)$account_category == "LeftTr
left_troll_on_the_issue = left_troll_on_the_issue/left_troll_total_words
```

```
## Save the results in a data frame for visualization:
results = data.frame(network = c("Left Troll", "Right Troll"),
                     prop_on_issue = c(left_troll_on_the_issue, right_troll_on_the_issue))
## Plot the results
ggplot(results, aes(x=reorder(network, -prop_on_issue), y=prop_on_issue))+
  geom_bar(stat="identity") + theme_minimal() + xlab("Partisan network") +
  ylab("Attention to issue (proportion)")+
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

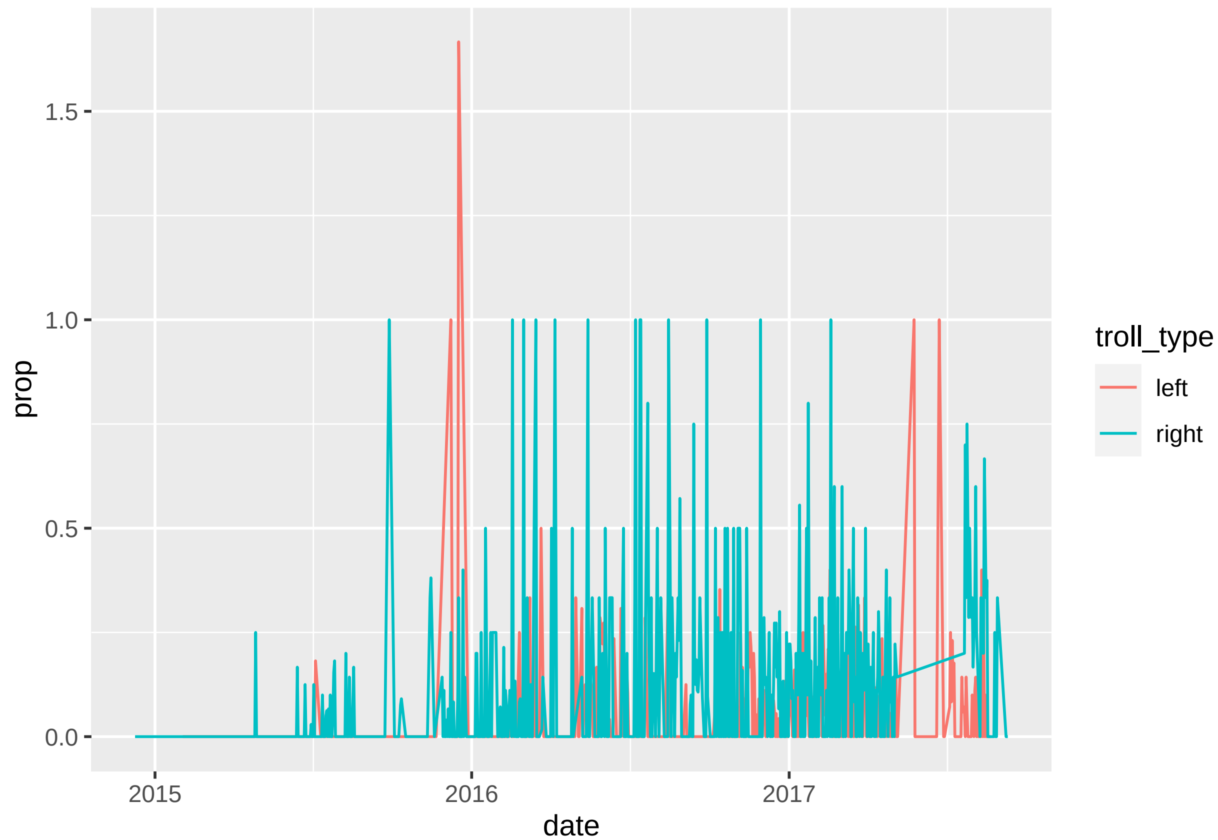


Right Troll accounts have a slightly higher proportion of their content dedicated to the issue compared to the Left Troll accounts, as indicated by the height of the bars. It shows that Russia IRA strategy on using more right troll on issue relate to Ukraine

Time Trend Plot

```
right_troll_on_the_issue = tokens_subset(russian_trolls_tokens, subset=account_category == "RightTroll"
  tokens_lookup(dictionary = ukraine_dict) %>%
  dfm()

right_prop_on_issue_overtime = aggregate(as.vector(right_troll_on_the_issue), by=list(docvars(right_tro
colnames(right_prop_on_issue_overtime) = c("date", "prop")
right_prop_on_issue_overtime$troll_type = "right"

left_troll_on_the_issue = tokens_subset(russian_trolls_tokens, subset=account_category == "LeftTroll")
  tokens_lookup(dictionary = ukraine_dict) %>%
  dfm()

left_prop_on_issue_overtime = aggregate(as.vector(left_troll_on_the_issue), by=list(docvars(left_troll_
colnames(left_prop_on_issue_overtime) = c("date", "prop")
left_prop_on_issue_overtime$troll_type = "left"

right_left_on_the_issue = rbind(right_prop_on_issue_overtime, left_prop_on_issue_overtime)
```

```r
ggplot(right_left_on_the_issue, aes(x=date, y=prop, group=troll_type)) +
  geom_line(aes(color=troll_type))
```



Sentiment Analysis

```r
## Load the dictionary:

load("/Users/minpan/Desktop/Text as Data/sentiment_dictionary.rdata")

## What are some "positive" words in the dictionary?
positive_words = sentiment_dictionary[sentiment_dictionary$y==1,]
sample(positive_words$x, 10)
```

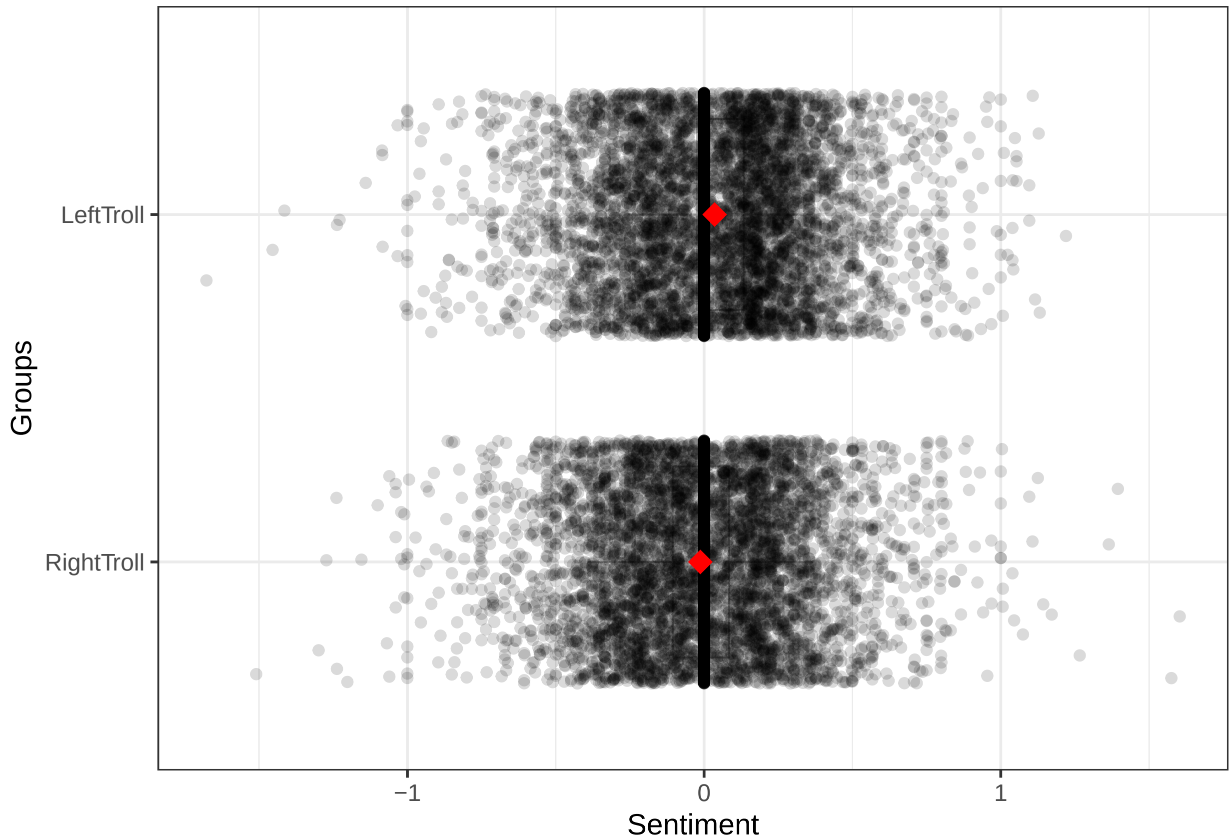
```
##  [1] "endearing"   "aspiring"    "redeem"      "justice"     "well wishers"
##  [6] "sincerity"   "thorough"    "bonny"       "gratifying"  "honor"
```

```r
## What are some "negative" words in the dictionary?
negative_words = sentiment_dictionary[sentiment_dictionary$y==-1,]
sample(negative_words$x, 10)
```

```
##  [1] "byzantine"     "pillory"       "polio"         "venomously"
##  [5] "acridly"       "wailing"       "annoyances"    "racism"
##  [9] "dissappointed" "oppositions"
```

Sentiment Score

```r
## The function get_sentences() tokenizes the free texts into sentences
russian_trolls_text = get_sentences(russian_trolls_sample$text)

## The function sentiment_by() calculates the sentiment of text by a grouping variable
russian_trolls_sentiment = sentiment_by(russian_trolls_text, by=russian_trolls_sample$account_category)
plot(russian_trolls_sentiment)
```



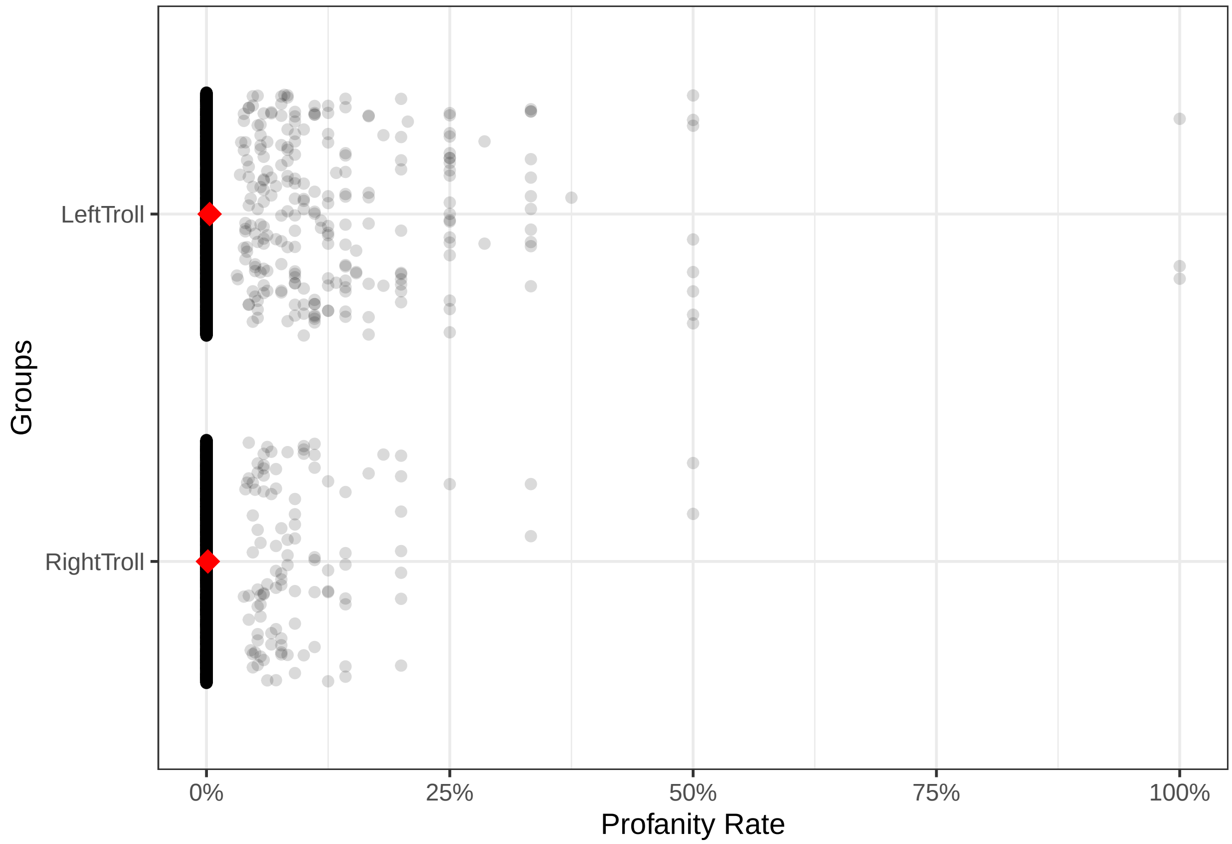
Tweets from "LeftTroll" accounts have a slightly positive average sentiment, whereas tweets from "RightTroll" accounts have a slightly negative average sentiment.

The standard deviation score indicates that "RightTroll" tweets's sentiment variability is slightly higher, which might suggest that they exhibit a broader range of sentiment than "LeftTroll" tweets.

Lastly, the plot shows that the differences in average sentiment scores between the two categories are not large, which might suggest that both groups tend to tweet content with a relatively neutral sentiment on average.

Profanity Analysis

```r
russian_trolls_profanity = profanity_by(russian_trolls_text, by=russian_trolls_sample$account_category)
plot(russian_trolls_profanity)
```



"LeftTroll" accounts use profanity more frequently on average than "RightTroll" accounts, as shown by the higher average profanity rate. The higher standard deviation for "LeftTroll" tweets shows they have more variability in the use of profanity than "RightTroll" tweets. However, even though "LeftTroll" accounts have a higher count and average of profanity, the overall rate is still low for both groups given the large total word counts.

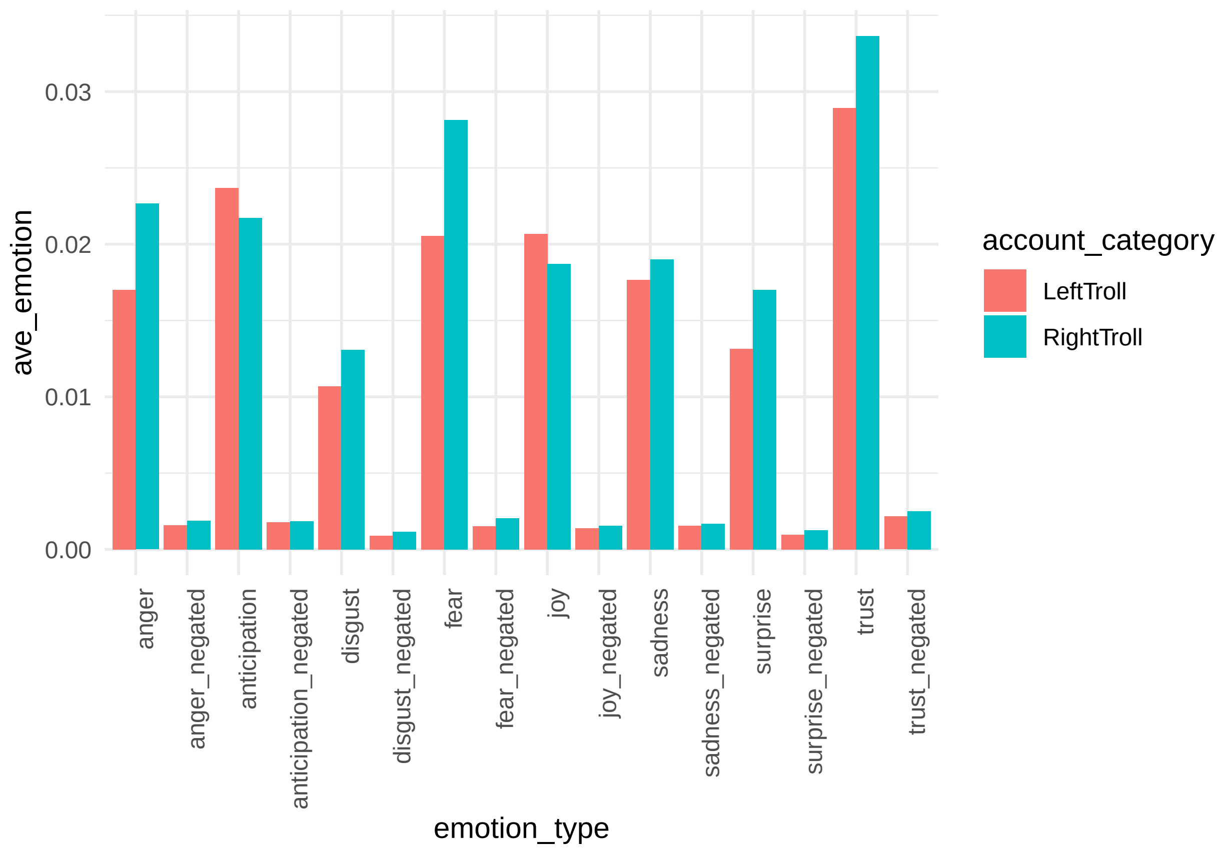
Emotion Analysis

```
## The function emotion_by() calculates the rate of emotion by a grouping variable (news outlet in our c
## The emotion score ranges between 0 (no emotion used) and 1 (all words used were emotional)
russian_troll_emotion = emotion_by(russian_trolls_text, by=russian_trolls_sample$account_category)
```

As shown by this plot, the tweets from the Right Troll group exhibit higher levels of emotion types, including anger, fear, disgust, sadness, surprise, and trust compared to the "Left Troll" group. Conversely, the Left Troll group displays higher levels of emotions related to anticipation and joy compared to the Right Troll group.

Overall, the Right Troll group seems to express more negative emotions in the tweets, whereas the Left Troll tweet group uses more positive emotional language.

```
## Let's plot the emotions. Who is most positive? Most negative?
ggplot(russian_troll_emotion, aes(x=emotion_type, y=ave_emotion, fill=account_category))+
  geom_bar(position="dodge",stat="identity") + theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



Machine Learning

Use machine learning models to classify the identity of the trolls based on the text that appears in their tweets.

The variable account_category includes the troll labels: RightTroll or LeftTroll.

Create training and test sets from this corpus, by randomly selecting 7,000 tweets to the training set and the remaining 3,000 to the test set. Create a DFM for each set and pre-process it.

install.packages(c("randomForest", "caret", "rpart", "rpart.plot"))

Classify the identity of the trolls based on the text that appears in their tweets.

```
russian_trolls_sample[1,]
```

```
##                  tweet_id          author
## 30717 695588727247237120 PRETTYLARAPLACE
##
## 30717 From the onset, Establishment knew  @realDonaldTrump has a long standing history as a #PATRIOT
##       account_category       date
## 30717       RightTroll 2016-02-05
```

```r
# Give numeric IDs to articles ( heloful for making training and test sets)
docvars(russian_trolls_corpus, "id_numeric") = 1:ndoc(russian_trolls_corpus)
head(docvars(russian_trolls_corpus))
```

```
##              tweet_id           author account_category       date id_numeric
## 1 695588727247237120 PRETTYLARAPLACE       RightTroll 2016-02-05          1
## 2 793549700947410944   REGIEBLACKMON        LeftTroll 2016-11-01          2
## 3 850534685222588416     SAMIRGOODEN        LeftTroll 2017-04-08          3
## 4 843906791314481152 PRETTYLARAPLACE       RightTroll 2017-03-20          4
## 5 836695892359065600   REGIEBLACKMON        LeftTroll 2017-02-28          5
## 6 857746120776314880    RAMONASNAILS        LeftTroll 2017-04-27          6
```

```r
#summary(russian_trolls_corpus)
```

Selecting 7,000 tweets to the training set and the remaining 3,000 to the test set. Create a DFM for each set and pre-process it.

```r
set.seed(12345)
# training set, save in the object "id_train":
id_train = sample(x=1:nrow(russian_trolls_sample), 7000, replace = FALSE)

# We use the corpus_subset() function to get the training set data from the corpus.
# We then create a DFM from this subset and do some preprocessing:
dfmat_training = corpus_subset(x= russian_trolls_corpus, subset = id_numeric %in% id_train) %>%
  tokens(remove_numbers = TRUE, remove_punct = TRUE, remove_url = TRUE,
         remove_symbols = TRUE) %>%
  tokens_select(pattern = stopwords('en'), selection = 'remove') %>%
  tokens_wordstem(language = "en") %>%
  tokens_tolower() %>%
  dfm() %>% dfm_tfidf()

# dimensions of training DFM?
dim(dfmat_training)
```

```
## [1]  7000 15093
```

```r
# Next, we get the test set by choosing all the articles that are NOT in the training set.
# We create a DFM for the test set:
dfmat_test = corpus_subset(russian_trolls_corpus, !id_numeric %in% id_train) %>%
  tokens(remove_numbers = TRUE, remove_punct = TRUE, remove_url = TRUE,
         remove_symbols = TRUE) %>%
  tokens_select(pattern = stopwords('en'), selection = 'remove') %>%
  tokens_wordstem(language = "en") %>%
  tokens_tolower() %>%
  dfm() %>% dfm_tfidf()

# What are the dimensions of our test DTM?
dim(dfmat_test)
```

```
## [1] 3000 8937
```

2. Trim

Trim the training and test DFMs using the dfm_trim() function. Set min_termfreq = 5 and max_termfreq = 400.

```
# Trim the training DFM
dfmat_training_trimmed <- dfm_trim(dfmat_training, min_termfreq = 5, max_termfreq = 400)
```

```
## Warning in dfm_trim.dfm(dfmat_training, min_termfreq = 5, max_termfreq = 400):
## dfm has been previously weighted
```

```
# Trim the test DFM
dfmat_test_trimmed <- dfm_trim(dfmat_test, min_termfreq = 5, max_termfreq = 400)
```

```
## Warning in dfm_trim.dfm(dfmat_test, min_termfreq = 5, max_termfreq = 400): dfm
## has been previously weighted
```

```
#  dimensions of the trimmed DFM
dim(dfmat_training_trimmed)
```

```
## [1] 7000 5696
```

```
dim(dfmat_test_trimmed)
```

```
## [1] 3000 3107
```

#Naive Bayes Classifier

Train a Naive Bayes classifier on the training set, and predict account_category in the test set.

The function textmodel_nb() classifies text using a Naive Bayes model. In the function, x is the DFM of the training data and y is the training labels associated with each training document:

```
classify_nb = textmodel_nb(x = dfmat_training, y = docvars(dfmat_training, "account_category"))
# We can access the class probabilities from the training set by using the code blow.
# We select just a subset for viewing:
dim(classify_nb$param)
```

```
## [1]     2 15093
```

```
classify_nb$param[,1:10]
```

```
##                    onset    establish         knew @realdonaldtrump         long
## LeftTroll   9.926696e-06 0.0001397008 0.0002268174     0.0002334697 0.0005046426
## RightTroll 4.956859e-05 0.0003312259 0.0001499383     0.0015845507 0.0003810415
##                    stand      histori     #patriot   #trumptrain    #trump2016
## LeftTroll   0.0005210940 0.0005639143 9.926696e-06 9.926696e-06 9.926696e-06
## RightTroll 0.0007688516 0.0004213158 4.956859e-05 5.307810e-04 4.198165e-04
```

terms like @realdonaldtrump, #trumptrain, and #trump2016 have higher probabilities for the RightTroll class, which suggests that these terms are more indicative of the RightTroll in training data.

```r
# The features with the highest probabilities of Left Troll
classify_nb$param[1,][order(classify_nb$param[1,], decreasing = T)][1:10]
```

```
##         amp       black         get       trump         now       white
## 0.002753921 0.002750079 0.002518939 0.002461254 0.002368574 0.002169306
##       peopl        like        just         one
## 0.002132984 0.002048255 0.001880511 0.001764689
```

```r
# The features with the highest probabilities of Right
classify_nb$param[2,][order(classify_nb$param[2,], decreasing = T)][1:10]
```

```
##       trump         new     hillari      #pjnet       #tcot       obama
## 0.003509222 0.002887001 0.002653261 0.002642325 0.002511613 0.002243195
##     clinton        want         say        like
## 0.001926710 0.001847829 0.001840578 0.001765853
```

1 refers to left, 2 refers to right as R orders factor levels alphabetically.

```r
# Naive Bayes can only consider words that occur both in the training set and the test set.
# Let's use the function dfm_match() to make the features identical
# in the training and test sets:
dfmat_matched = dfm_match(dfmat_test, features = featnames(dfmat_training))

# What are the dimensions of our matched test set?
dim(dfmat_matched)
```

```
## [1]  3000 15093
```

```r
# Examine how this compared to the original dimensions of our test set:
dim(dfmat_test)
```

```
## [1] 3000 8937
```

-any features (words) that were not present in the training set are removed from the test set, and any features that are in the training set but not in the test set are added with a frequency of zero.

Calculate Confusion Matrix and Metrics

```r
## Make Predictions on the Test Set:
predicted_class = predict(classify_nb, newdata = dfmat_matched, type="class")

## To evaluate performance, we examine the confusion matrix:
## confusionMatrix() is a powerful function from the 'caret' package that provides information
## on many model performance metrics:
actual_class = docvars(dfmat_matched, "account_category")

tab_class = table(predicted_class, actual_class)

confusionMatrix(tab_class, mode = "everything")
```

```
## Confusion Matrix and Statistics
##
##                actual_class
## predicted_class LeftTroll RightTroll
##      LeftTroll       1141        305
##      RightTroll       436       1118
##
##                Accuracy : 0.753
##                  95% CI : (0.7372, 0.7683)
##     No Information Rate : 0.5257
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5069
##
##  Mcnemar's Test P-Value : 1.791e-06
##
##             Sensitivity : 0.7235
##             Specificity : 0.7857
##          Pos Pred Value : 0.7891
##          Neg Pred Value : 0.7194
##               Precision : 0.7891
##                  Recall : 0.7235
##                      F1 : 0.7549
##              Prevalence : 0.5257
##          Detection Rate : 0.3803
##    Detection Prevalence : 0.4820
##       Balanced Accuracy : 0.7546
##
##        'Positive' Class : LeftTroll
##
```

```r
confusionMatrix(tab_class, mode = "everything", positive="RightTroll")
```

```
## Confusion Matrix and Statistics
##
##                actual_class
## predicted_class LeftTroll RightTroll
##      LeftTroll       1141        305
##      RightTroll       436       1118
##
##                Accuracy : 0.753
##                  95% CI : (0.7372, 0.7683)
##     No Information Rate : 0.5257
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5069
##
##  Mcnemar's Test P-Value : 1.791e-06
##
##             Sensitivity : 0.7857
##             Specificity : 0.7235
##          Pos Pred Value : 0.7194
##          Neg Pred Value : 0.7891
##               Precision : 0.7194
```

```
##                  Recall : 0.7857
##                      F1 : 0.7511
##              Prevalence : 0.4743
##          Detection Rate : 0.3727
##    Detection Prevalence : 0.5180
##       Balanced Accuracy : 0.7546
##
##        'Positive' Class : RightTroll
##
```

4. Decision Tree Classifier

Train a classification tree on the training set, and predict account_category in the test set.

```r
# To classify fake/real news with a decision tree, we will work with the rpart() function.

# The function take a data frame as an input, so we'll convert our DFMs from quanteda
# objects into data frames using the convert() function:
dfmat_training_df = convert(dfmat_training, to="data.frame")
dfmat_test_df = convert(dfmat_matched, to="data.frame")

# Now, we'll add the labels to our training data frame:
dfmat_training_df$partisan_label <- as.factor(docvars(dfmat_training, "account_category"))

# The funciton rpart() will fit a decision tree into our training data:
classify_tree = rpart(formula = partisan_label ~ ., data=dfmat_training_df[,!colnames(dfmat_training_df)

## Let's predict the classes in the test set using the predict() function:
predicted_class = predict(classify_tree, newdata = dfmat_test_df[,!colnames(dfmat_test_df) == "doc_id"]

## Evaluate performance with the confusion matrix:
tab_class = table(predicted_class, actual_class)
confusionMatrix(tab_class, mode = "everything")
```

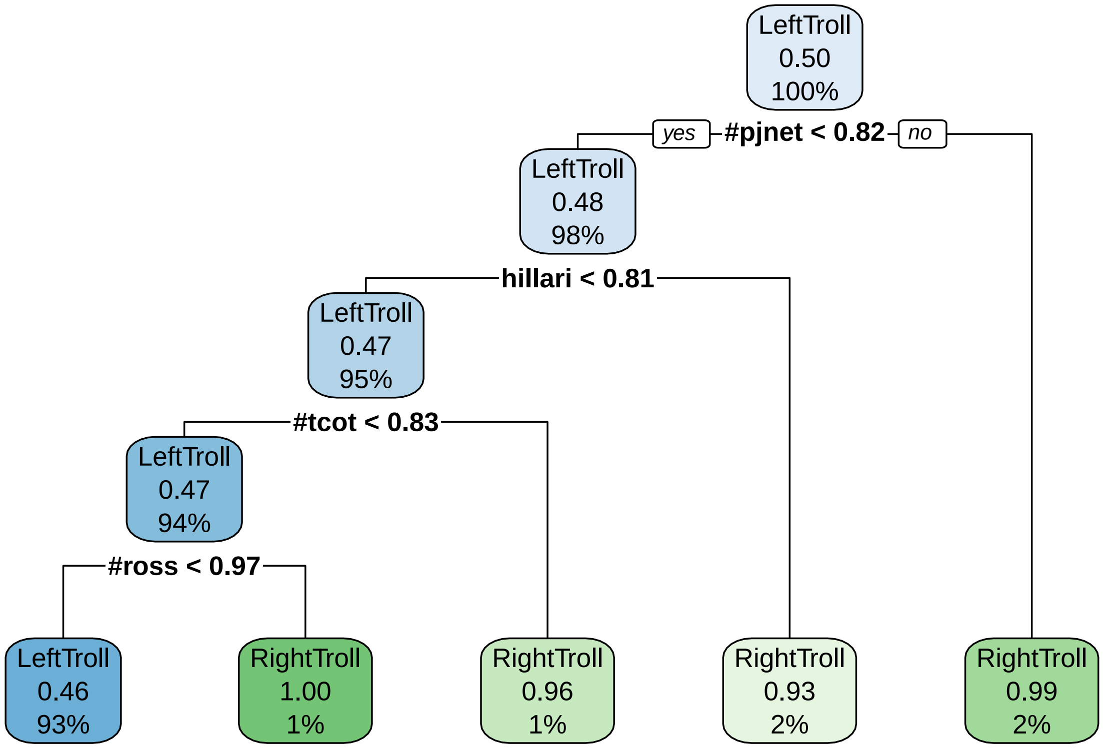
```
## Confusion Matrix and Statistics
##
##                actual_class
## predicted_class LeftTroll RightTroll
##      LeftTroll       1565       1237
##      RightTroll        12        186
##
##                Accuracy : 0.5837
##                  95% CI : (0.5658, 0.6014)
##     No Information Rate : 0.5257
##     P-Value [Acc > NIR] : 9.889e-11
##
##                   Kappa : 0.1285
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9924
##             Specificity : 0.1307
```

```
##          Pos Pred Value : 0.5585
##          Neg Pred Value : 0.9394
##               Precision : 0.5585
##                  Recall : 0.9924
##                      F1 : 0.7148
##              Prevalence : 0.5257
##          Detection Rate : 0.5217
##    Detection Prevalence : 0.9340
##       Balanced Accuracy : 0.5616
##
##        'Positive' Class : LeftTroll
##
```

```r
confusionMatrix(tab_class, mode = "everything", positive = "RightTroll")
```

```
## Confusion Matrix and Statistics
##
##                actual_class
## predicted_class LeftTroll RightTroll
##      LeftTroll       1565       1237
##      RightTroll        12        186
##
##                Accuracy : 0.5837
##                  95% CI : (0.5658, 0.6014)
##     No Information Rate : 0.5257
##     P-Value [Acc > NIR] : 9.889e-11
##
##                   Kappa : 0.1285
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.1307
##             Specificity : 0.9924
##          Pos Pred Value : 0.9394
##          Neg Pred Value : 0.5585
##               Precision : 0.9394
##                  Recall : 0.1307
##                      F1 : 0.2295
##              Prevalence : 0.4743
##          Detection Rate : 0.0620
##    Detection Prevalence : 0.0660
##       Balanced Accuracy : 0.5616
##
##        'Positive' Class : RightTroll
##
```

```r
## We can visualize our tree with the function rpart.plot()
## Each node shows:
# - the predicted class
# - the predicted probability of fake,
# - the percentage of observations in the node

rpart.plot(classify_tree, roundint=F)
```



Performance Comparison and Evaluation

Accuracy: The Naive Bayes classifier has a significantly higher accuracy (75.3%) compared to the classification tree (58.37%);

Precision and Recall Balance: The Naive Bayes classifier has a better balance between precision and recall for both classes. However, for the classification tree, it shows a high recall but very low precision for 'LeftTroll', and high precision but very low recall for 'RightTroll'.

F1 Score: The F1 score of Naive Bayes classifier is considerably higher for both 'LeftTroll' and 'RightTroll', which shows a better balance between precision and recall. Whereas, the classification tree has a very low F-1 score for 'RightTroll'.

Conclusion:

The Naive Bayes classifier performed better than the classification tree across all major performance metrics. It has a higher accuracy, a more balanced precision and recall, and a higher F1 score for both troll types. Whereas the classification tree showed a bias towards one class ('LeftTroll') and struggled to identify 'RightTroll' accurately, as indicated by its very low recall for 'RightTroll'. Therefore, based on the model performance statistics, the Naive Bayes classifier did a relatively good job of predicting the type of trolls and outperformed the classification tree.