Fake News Classifier

A Data Science Project

Marcel Arnold (1710389)

4/23/2018

Contents

Problem Statement & Dataset Choice

The latest hot topic in the news is fake news and many are wondering what data scientists can do to detect it, find the underlying patterns and maybe prevent it. The used Kaggle dataset contains text and metadata scraped from 244 websites, contains news articles published between 26/10/2016 and 25/11/2016 and is mostly related to the US presidential campaign in 2016, and the tagging is based on the curated list of www.opensources.co

Dataset source: www.kaggle.com/mrisdal/fake-news

Tags

Open Sources uses combinations of the following tags to classify each website we assess.

Fake Sources that entirely fabricate information, disseminate deceptive content, or grossly distort actual news reports

Satire Sources that use humor, irony, exaggeration, ridicule, and false information to comment on current events.

Bias Sources that come from a particular point of view and may rely on propaganda, decontextualized information, and opinions distorted as facts.

Conspiracy Sources that are well-known promoters of kooky conspiracy theories.

Rumor Sources that traffic in rumors, gossip, innuendo, and unverified claims.

State Sources in repressive states operating under government sanction.

Junksci Sources that promote pseudoscience, metaphysics, naturalistic fallacies, and other scientifically dubious claims.

Hate Sources that actively promote racism, misogyny, homophobia, and other forms of discrimination.

Clickbait Sources that provide generally credible content, but use exaggerated, misleading, or questionable headlines, social media descriptions, and/or images.

Unreliable Sources that may be reliable but whose contents require further verification.

Political Sources that provide generally verifiable information in support of certain points of view or political orientations.

Reliable Sources that circulate news and information in a manner consistent with traditional and ethical practices in journalism.

Objectives & Methodology

The main objective of this coursework project is to analyse a text-based, labelled dataset in a quantitative way in order to build a machine learning model which is able to distinguish between different classes of news articles according to a given set of labels. These (11) labels are pre-coded, and I will propably recluster them to get better results. Furthermore, I will look into textual associations, visualise the text data by different features (by label, by frequency and more) and by different means (tables, diagrams, wordclouds) during and after I've applied several R text mining methods. After cleaning, transforming and filtering the text data into numerical data I will train and test several Naive Bayes based models. Finally, I am going to compare and validate the performances of these models with more sophisticated unsupervised machine learning algorithms.

0. Requirements

At first, I need lots of R libraries, which I will pre-load in order to maintain structure.

```r
library(ggplot2)
# visualising data
library(caret)
# The caret package (short for _C_lassification _A_nd _RE_gression _T_raining) is a set
# of functions that attempt to streamline the process for creating predictive models.
library(SnowballC)
# A library that implements a word stemming algorithm for collapsing words to a
# common root to aid comparison of vocabulary
```

```r
library(RCurl) # reading data from external web sources
library(tm) # big library for text mining tools
library(e1071) # contains several machine learning functions, e.g. Naive Bayes
library(klaR) # another library with miscellaneous functions for classification
# and visualization
library(plyr) # Tools for Splitting, Applying and Combining Data
library(wordcloud) # creating word clouds
library(qdap) # package for cleaning and analysing text data
library(wordnet) # package for stemCompletion (didnt really work)
#library(RWeka) # huge libray for machine learning applications in R.
#I am using especially the tokenizing function.
# Won't initialise it due to perfomance issues

# setting working directory
dir <- getwd()
setwd(dir)

# loading datasets
if(!exists("fnews")) {
  fnews <- read.csv("fake.csv", header = TRUE)
  sources <- read.csv(text=
getURL("https://raw.githubusercontent.com/BigMcLargeHuge/opensources/master/sources/sources.csv"))
}
```

Let's show a first overview of the labels:

```r
table(fnews$type)
```

```
##
##       bias         bs conspiracy       fake       hate    junksci
##        443      11492        430         19        246        102
##     satire      state
##        146        121
```

Type 'bs' means that the date is unlaballed, which is why I have to re-label it with the original label source ([www.OpenSources.com]).

1. Data pre-processing

1.1 Merging and Cleansing

After matching the fnews dataset with the OpenSources list I will have prepare the dataset a little bit more:

- shuffle dataset so that the labels are almost equally distributed
- removing unneccessary columns and rows
- transform to text data to character and UTF-8 and factorise label column

```r
# match fnews with open sources and get 'types'
df1 <- merge(fnews, sources, by.x="site_url", by.y="X")

# randomly sort (shuffle) to achieve better proportions for splitting later
set.seed(23)
df2 <- df1[sample(nrow(df1)),]

# rename type columns
```

```r
df2$type1 <- as.factor(df2$type.y)

# remove unneccessary columns
drops <- c("uuid","type.x","X.1","Source.Notes..things.to.know..","ord_in_thread","type.y",
           "X2nd.type","X3rd.type", "main_img_url", "spam_score","replies_count",
           "participants_count", "likes", "comments", "shares", "thread_title",
           "country", "crawled")
data <- df2[ , !(names(df2) %in% drops)]

#  factorise labels
data$type1 <- tolower(data$type1)
data$type1 <- factor(data$type1)

# remove non-english instances
data <- data[(data$language=="english"),]
data$language = NULL

# remove rows with N/A
rname <- names(data)[which.max(sapply(data, function(x) sum(is.na(x))))]
data <- data[ , !(names(data) == rname)]

# concatenate title and article vector
data <- within(data,  text <- paste(title, text, sep=" "))
data$title = NULL

# try UTF-8 encoding
table(Encoding(data$text))
```

```
##
## unknown
##   10988
```

```r
data$text <- as.character(data$text)
data$text <- enc2utf8(data$text)
table(Encoding(data$text))
```

```
##
## unknown   UTF-8
##    1507    9481
```

```r
#refactor class labels
data$type1 <- factor(data$type1)
original_labels <- as.character(data$type1)

# Labels are now fixed:
table(data$type1)
```

```
##
##       bias  clickbait conspiracy       fake       hate    junksci
##       2561        775       2486        651        677        643
##  political   reliable      rumor     satire unreliable
##        972         93        107       1020       1003
```

```r
# remove unnecessary variables from environment
rm(drops, df1, df2, rname)
```

There are still > 1.000 rows which are not UTF-8 encoded, therefore contain "native or non-ASCII" characters. Can't fix that at the moment - but these issues will be removed during the text pre-processing phase.

1.2 First Data Exploration

In total, the dataset has approx. 11.000 entries, 11 tags and 5 variables (from which I will most likely just use "text" to create the indicator features and "type1" as labels).

```r
print("### Rows ###")
```

```
## [1] "### Rows ###"
```

```r
dim(data)[1]
```

```
## [1] 10988
```

```r
print("### Columns ###")
```

```
## [1] "### Columns ###"
```

```r
dim(data)[2]
```

```
## [1] 5
```

```r
print("### Levels of label ###")
```

```
## [1] "### Levels of label ###"
```

```r
levels(data$type1)
```

```
##  [1] "bias"       "clickbait"  "conspiracy" "fake"       "hate"
##  [6] "junksci"    "political"  "reliable"   "rumor"      "satire"
## [11] "unreliable"
```

```r
print("### Column names ###")
```

```
## [1] "### Column names ###"
```

```r
colnames(data)
```

```
## [1] "site_url"  "author"    "published" "text"      "type1"
```

But we can see, that the label "bs" disappeared. Unfortunately, we just got 93 **reliable** items. All other rows seem to be labelled as **misleading** articels.

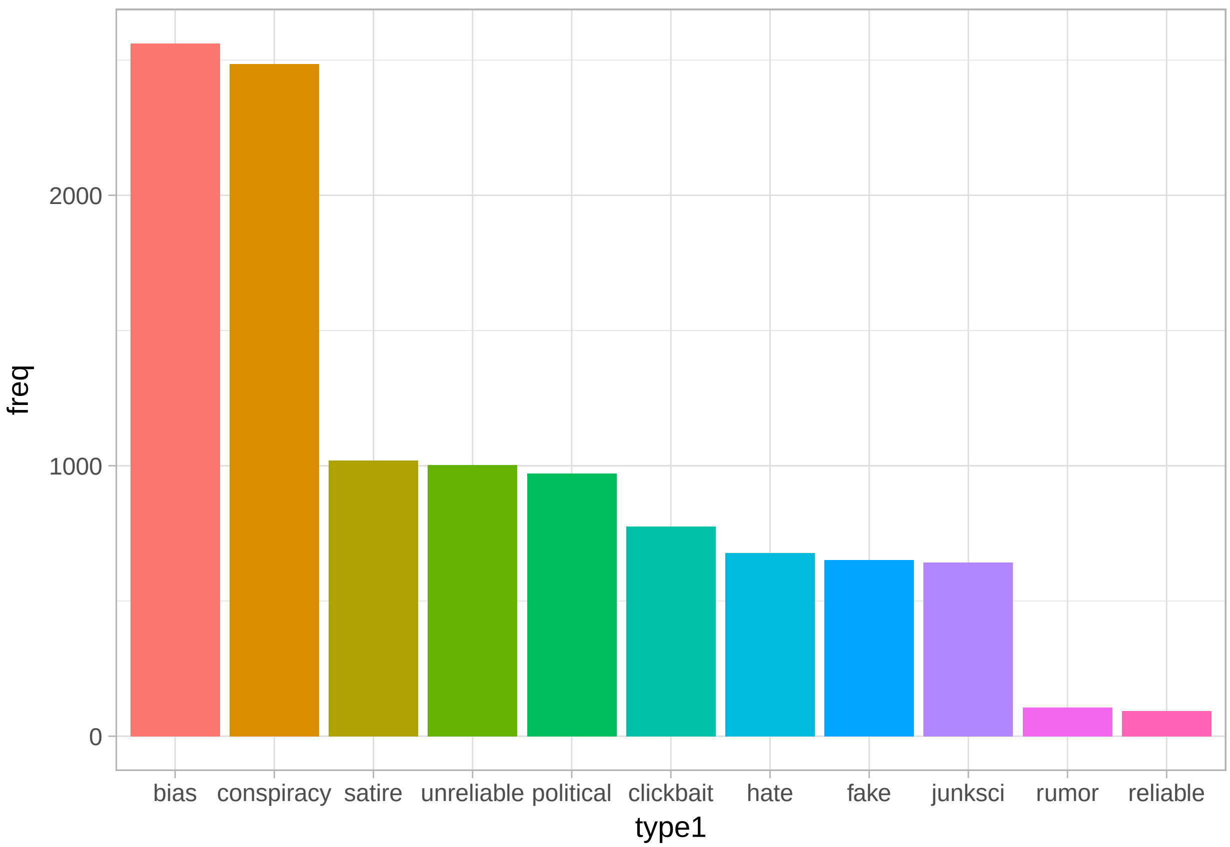
To get a better understanding of the labels, I will plot them in a bar chart, sorted by frequency:

```r
# plotting the labels
x <- count(data, 'type1')
x <- x[order(-x$freq),]
x$type1 <- factor(x$type1, levels = x$type1[order(-x$freq)])
x
```

```
##         type1 freq
## 1        bias 2561
## 3  conspiracy 2486
## 10     satire 1020
## 11 unreliable 1003
## 7   political  972
## 2   clickbait  775
## 5        hate  677
```

```
## 4        fake  651
## 6     junksci  643
## 9       rumor  107
## 8    reliable   93
```

```
## bar chart
g <- ggplot(x, aes(x=type1, y=freq, fill=type1))
g <- g + geom_bar(stat = "identity") + theme_light() + guides(fill=FALSE)
g
```




```
rm(g,x) # remove plot variables
```

2. Text Mining

The first step towards creating the classifier involves processing the raw text data for analysis. Text data are challenging to prepare because it is necessary to transform the terms and sentences into a numerical form that a computer can read. I will convert the data into something known as bag-of-words, which ignores the order of appearance of the words and simply provides a variable indicating whether the word appears (and how often). (mctear)

2.1 Preparing corpus creation

Before starting heavy computational calculations, I am going to load the "parallel" package to ensure that R is using the CPU cores efficiently.

```r
library(parallel)
options(mc.cores=1)
```

Now, the text data will be prepared for the corpus creation:

- replacing characters
- removing text between brackets
- convert abbreviations to full words
- convert symbols to words

```r
data$text <- gsub(data$text, pattern = "’", replacement = "'")
data$text <- gsub(data$text, pattern = "-", replacement = " ")
pre_corpus <- bracketX(data$text) # remove content within brackets
pre_corpus <- replace_abbreviation(pre_corpus) # convert e.g. "Mr." to "Mister"
#pre_corpus <- replace_contraction(pre_corpus)
# convert "isn't" to "is not" -> didn't really work
pre_corpus <- replace_symbol(pre_corpus) # convert $ to dollar
```

2.2 Creating Corpus

Before running several text mining functions, I need to transform the articles to a collection of text documents, a so-called "Corpus". A VCorpus in tm refers to "Volatile" corpus which means that the corpus is stored in memory and would be destroyed when the R object containing it is destroyed.

((https://stats.stackexchange.com/users/86949/indi), n.d.)

```r
dfCorpus = VCorpus(VectorSource(pre_corpus))
rm(pre_corpus)
dfCorpus[["20"]][["content"]] # show example entry
```

```
## [1] "New Report Uncovers Secret Trump Server That Repeatedly Communicated With Russia By Sean Colaro
```

2.3 Clean corpus

In order to analyse the text data I need to apply further functions on the corpus:

- Transform to lowercase, remove punctuations and numbers
- Removing standard stopwords (e.g. 'for', 'my' etc.) and own stopwords (URL parts in this case)
- Removing special characters
- Stemming words (e.g. reduced > reduc / computer, computational > compute)
- *(Lemmatizing: Method to reverse the stemming and get syntactically and better readable words. I've tried that with the "wordnet" package, but did not achieve workable results.)*
- Replace contractions with full words
- Strip whitespace

```r
dfCorpus <- tm_map(dfCorpus, content_transformer(tolower)) # transform to lowercase
dfCorpus <- tm_map(dfCorpus, removeNumbers) # remove all numbers
dfCorpus <- tm_map(dfCorpus, removePunctuation) # remove punctuation
```

```r
# replace common contractions
dfCorpus<-tm_map(dfCorpus, content_transformer(function(x)
  gsub(x,pattern="'re",replacement=" are")))
dfCorpus<-tm_map(dfCorpus, content_transformer(function(x)
  gsub(x,pattern="'m",replacement=" am")))
dfCorpus<-tm_map(dfCorpus, content_transformer(function(x)
```

```
  gsub(x,pattern="'ve",replacement=" have")))
dfCorpus<-tm_map(dfCorpus, content_transformer(function(x)
  gsub(x,pattern="n't",replacement=" not")))

# remove other special characters
replace_list <- c("/", "@", "\\|", "\"", """, """, "-", "-", "‘", "_", "'s")

for (r in replace_list) {
  dfCorpus <- tm_map(dfCorpus, content_transformer(function(x)
    gsub(x,pattern=r,replacement="  ")))
}
rm(r, replace_list)
```

```
dfCorpus <- tm_map(dfCorpus, content_transformer(function(x)
  gsub(x, pattern = "hillari",replacement = "hillary")))
dfCorpus <- tm_map(dfCorpus, removeWords, c(stopwords("english"),
                                             "http","https","www","dot","com","pictwitt"))
dfCorpus <- tm_map(dfCorpus, stripWhitespace)
```

```
# stemming
dfCorpus <- tm_map(dfCorpus, stemDocument)
```

The corpus is now cleaned and prepared:

```
dfCorpus[["20"]][["content"]]
```

```
## [1] "new report uncov secret trump server repeat communic russia sean colarossi mon oct st pm media
```

2.4 Convert to Document Term Matrix

Now, the corpus is prepared and can be converted to a document term matrix.

> A document-term matrix or term-document matrix is a mathematical matrix that describes the frequency of terms that occur in a collection of documents. In a document-term matrix, rows correspond to documents in the collection and columns correspond to terms.

(Wikipedia 2018b)

Simple DTM:

Documents	Term1	Term2	Term3
document1	0	0	1
document2	1	0	0
document3	0	1	0

For my text analysis I am especially interested in terms that appear frequently together. Therefore, I am not extracting the most frequent single words, but word pairs for which I will need to use a *n-gram tokenizer*. In my case a bi-gram tokenizer:

```
bigram_tokenizer <- function(x) {
  RWeka::NGramTokenizer(x, RWeka::Weka_control(min=2, max=2))}

dtm <- DocumentTermMatrix(dfCorpus,control=list(tokenize = bigram_tokenizer))
# Just in case, I will also generate a Term Document Matrix.
tdm <- TermDocumentMatrix(dfCorpus,control=list(tokenize = bigram_tokenizer))
```

```r
save.image("~/Dropbox/Studies/Semester 02/Data Science Development/coursework/fake_news/fnews.RData")
```

2.5 Sparsity

We now want to remove terms with a low single frequency but collectively huge amount of appearances.

```r
# Removing Sparse Terms
dim(dtm)
```

```
## [1]   10988 1643067
```

```r
dtms <- removeSparseTerms(dtm, 0.98)
dim(dtms)
```

```
## [1] 10988   146
```

```r
# frequencies of frequencies
freq <- colSums(as.matrix(dtms))
head(table(freq), 15)
```

```
## freq
## 230 235 239 243 248 250 252 254 261 265 268 269 273 279 286
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   2
```

```r
tail(table(freq), 15)
```

```
## freq
##  966  990 1010 1018 1047 1196 1210 1275 1293 1576 1978 2119 4244 5608 7865
##    1    2    1    1    1    1    1    1    1    1    1    1    1    1    1
```

As we can see, I was able to reduce the list of features from approx. 1.6 mio to 146.

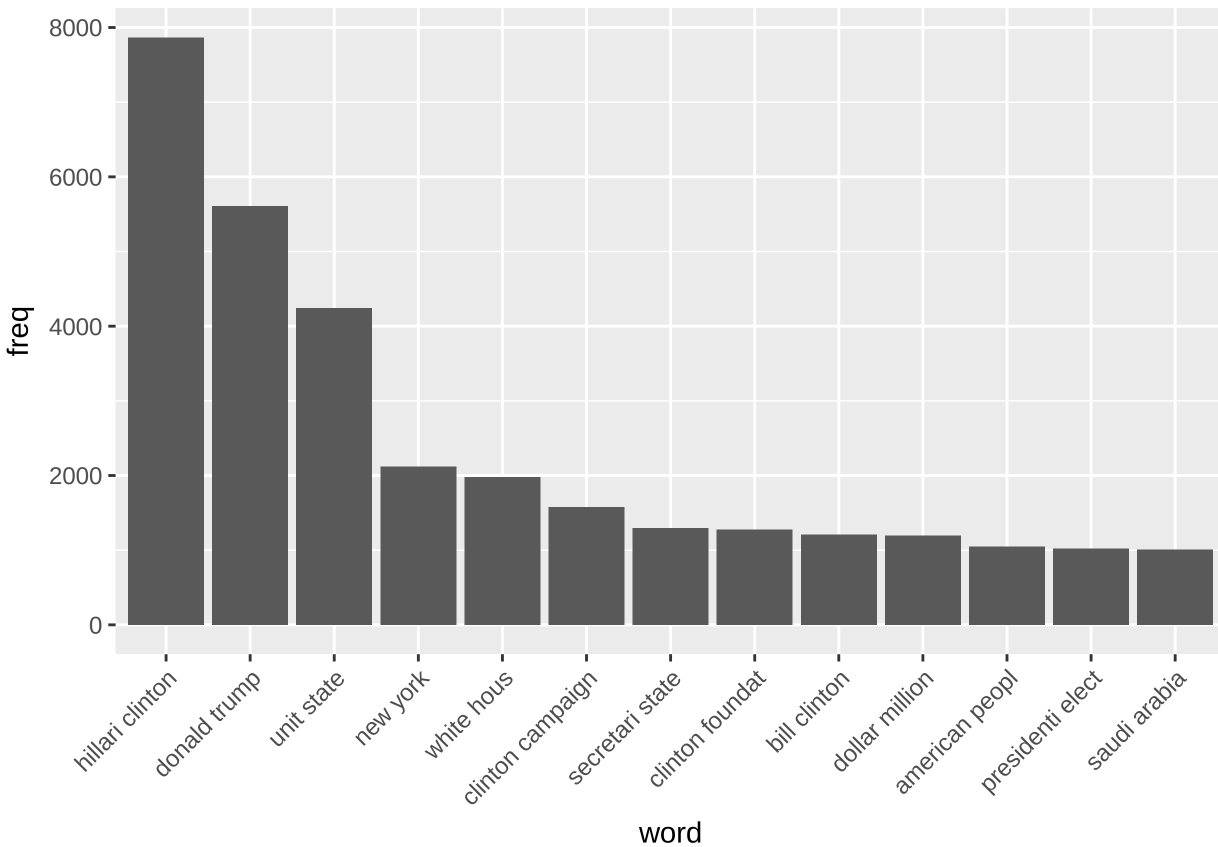
2.6 Plotting Word Frequencies

```r
freq <- sort(colSums(as.matrix(dtms)), decreasing=TRUE)
wf <- data.frame(word=names(freq), freq=freq)
head(wf,20)
```

```
##                                 word freq
## hillari clinton   hillari clinton 7865
## donald trump         donald trump 5608
## unit state             unit state 4244
## new york                 new york 2119
## white hous             white hous 1978
## clinton campaign clinton campaign 1576
## secretari state   secretari state 1293
## clinton foundat   clinton foundat 1275
## bill clinton         bill clinton 1210
## dollar million     dollar million 1196
## american peopl     american peopl 1047
## presidenti elect presidenti elect 1018
## saudi arabia         saudi arabia 1010
## barack obama         barack obama  990
## foreign polici     foreign polici  990
## year ago                 year ago  966
```

```
## middl east             middl east  932
## mainstream media mainstream media  923
## wall street           wall street  896
## democrat parti     democrat parti  870
```

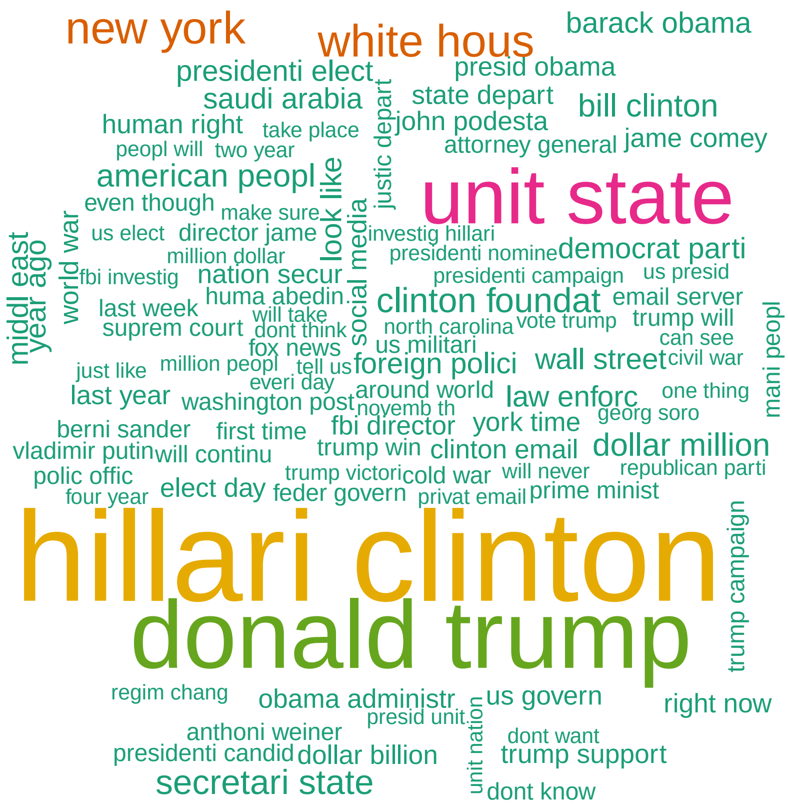
```
wf$word <- factor(wf$word, levels = wf$word[order(-wf$freq)])
wf_plot <- subset(wf, freq>1000)

g <- ggplot(wf_plot, aes(word, freq)) + geom_bar(stat="identity")
g <- g + theme(axis.text.x=element_text(angle=45, hjust=1))
g
```



2.7 Word Clouds

A much better visualisation of text content and their frequencies are word clouds:

```
set.seed(23)
wordcloud(names(freq), freq, max.words=100, colors=brewer.pal(6, "Dark2"))
```



2.8 Word Clouds per label

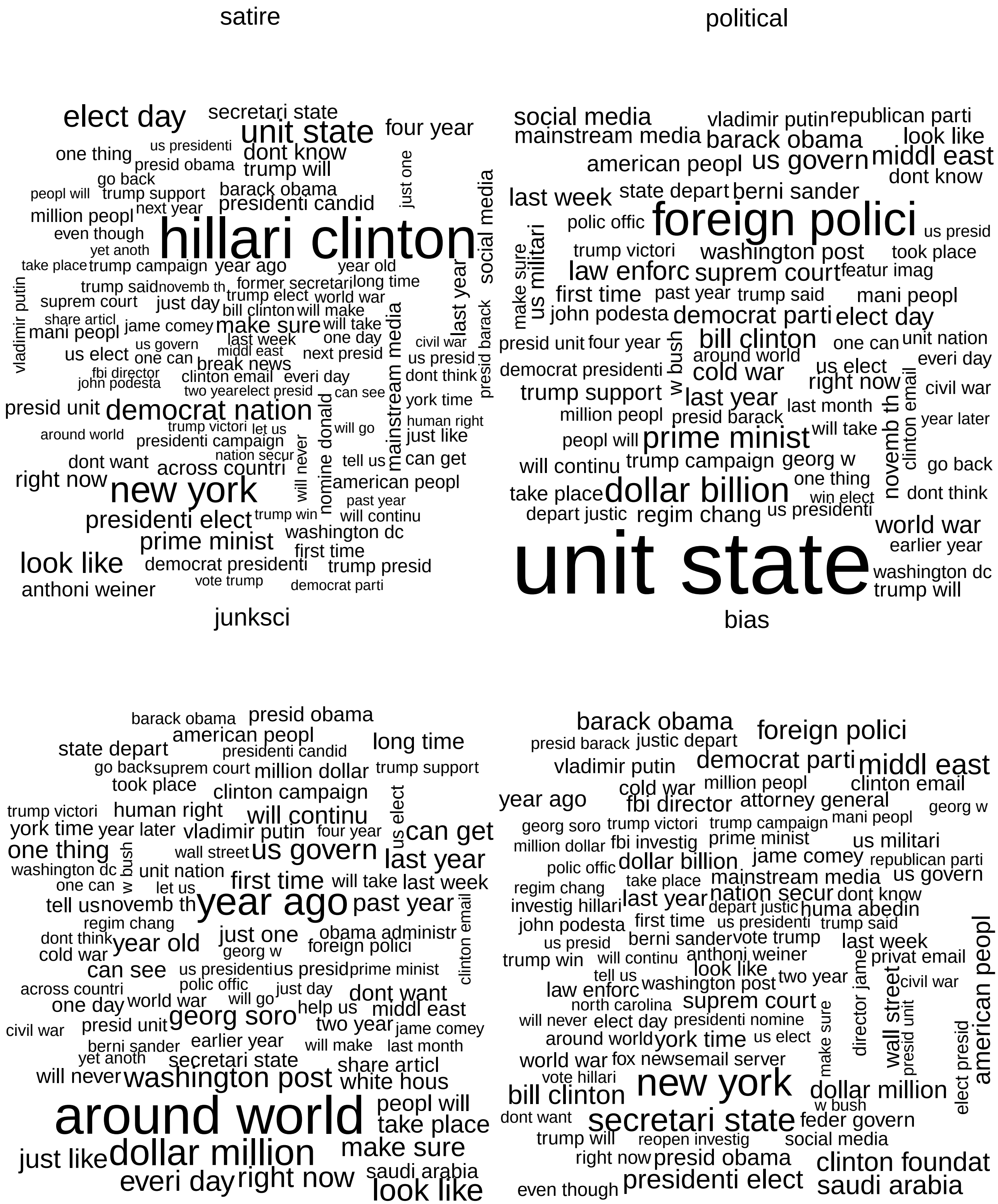
We get a better look, if we create wordclouds for each label:

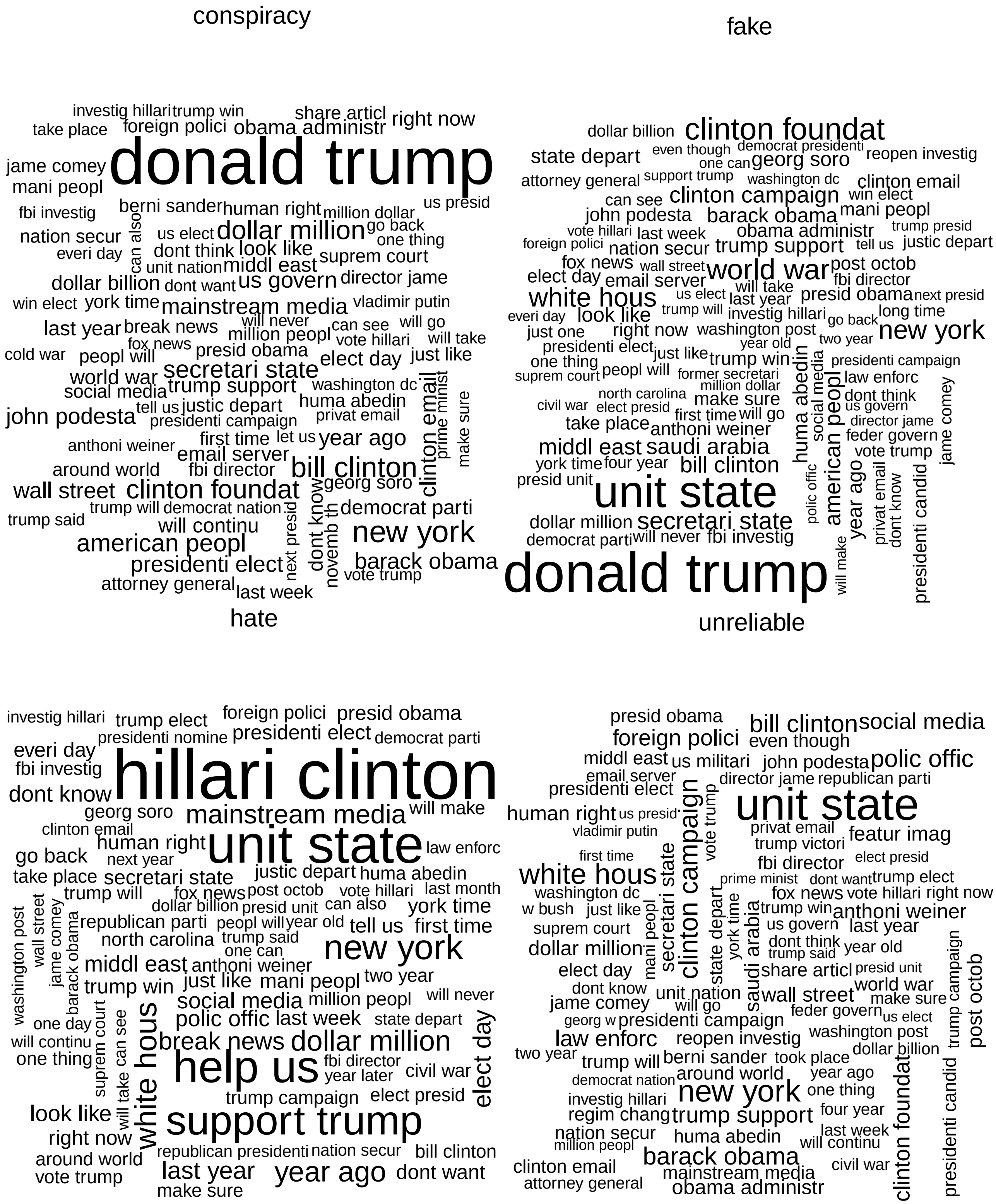
```
set.seed(23)

#Attach classification to each row
wc <- as.matrix(dtms)
wc <- cbind(wc, c(0, 1))
colnames(wc)[ncol(wc)] <- 'y'
wc <- as.data.frame(wc)
colnames(wc)[ncol(wc)]
```

```
## [1] "y"
```

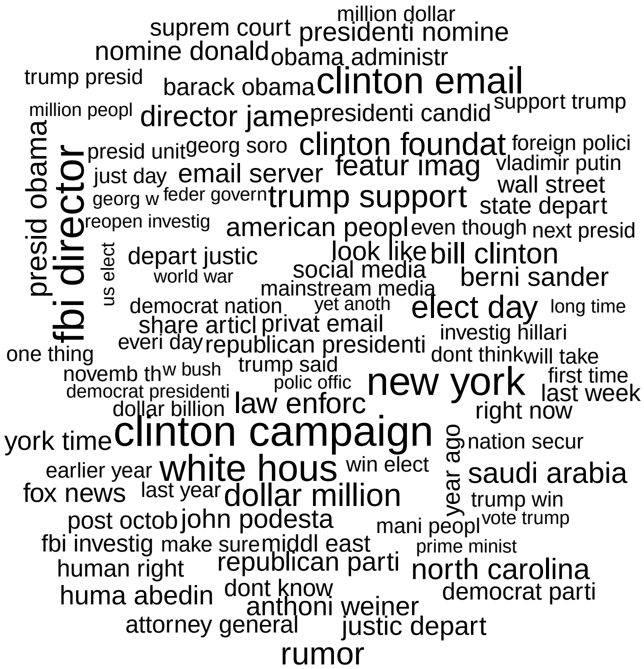
```
wc$y <- as.factor(data$type1)

labels <- unique(wc$y)

for (l in labels) {
  x <- subset(wc, y == l)
  freq <- sort(colSums(as.matrix(x[,-ncol(x)])), decreasing=TRUE)
  wf <- data.frame(word=names(freq), freq=freq)

  layout(matrix(c(1, 2), nrow=2), heights=c(1, 4))
  par(mar=rep(0, 4))
  plot.new()
  text(x=0.5, y=0.5, l)
```

```
  wordcloud(names(freq), freq, max.words=100)
}
```

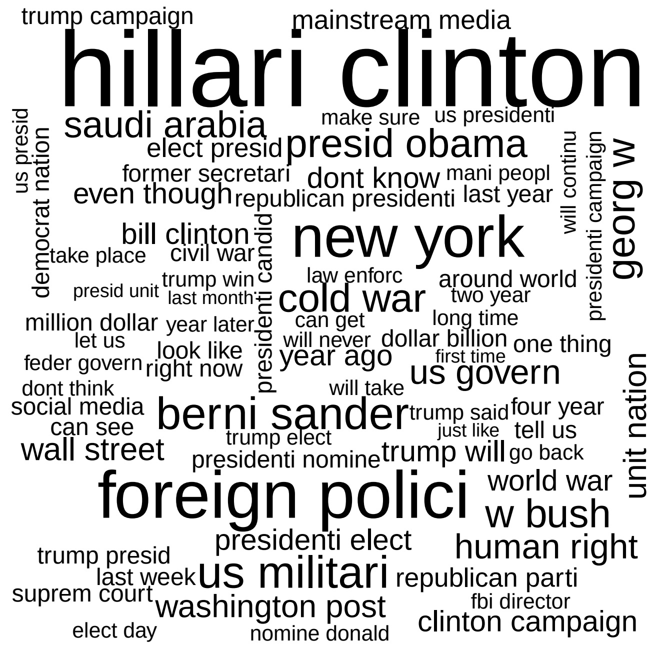


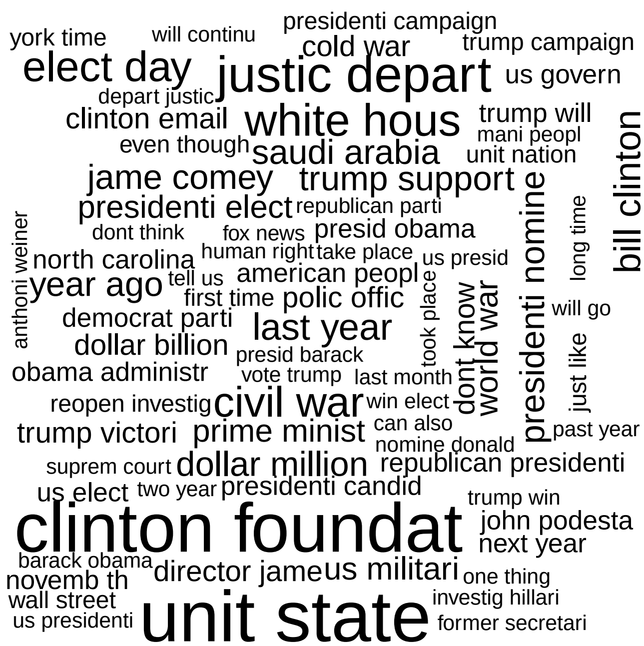

conspiracy
donald trump
investig hillari
trump win
share articl
right now
take place
foreign polici
obama administr
jame comey
mani peopl
fbi investig
berni sander
human right
million dollar
us presid
nation secur
us elect
dollar million
go back
one thing
everi day
can also
dont think
look like
suprem court
unit nation
middl east
dollar billion
dont want
us govern
director jame
win elect
york time
mainstream media
vladimir putin
last year
break news
will never
million peopl
can see
will go
cold war
fox news
peopl will
vote hillari
will take
presid obama
elect day
just like
world war
secretari state
social media
trump support
washington dc
prime minist
make sure
john podesta
tell us
justic depart
huma abedin
clinton email
presidenti campaign
privat email
anthoni weiner
first time
let us
year ago
email server
around world
fbi director
bill clinton
wall street
clinton foundat
georg soro
trump will
democrat nation
democrat parti
trump said
will continu
next presid
dont know
novemb th
new york
american peopl
presidenti elect
barack obama
vote trump
attorney general
last week
fake
dollar billion
clinton foundat
state depart
even though
democrat presidenti
one can
georg soro
reopen investig
attorney general
support trump
washington dc
clinton email
can see
clinton campaign
win elect
john podesta
barack obama
mani peopl
vote hillari
last week
obama administr
trump presid
foreign polici
nation secur
trump support
tell us
justic depart
fox news
wall street
world war
post octob
elect day
email server
will take
fbi director
white hous
us elect
last year
presid obama
next presid
everi day
look like
trump will
investig hillari
go back
long time
just one
right now
washington post
two year
new york
presidenti elect
year old
one thing
just like
trump win
huma abedin
presidenti campaign
suprem court
peopl will
former secretari
social media
law enforc
north carolina
million dollar
make sure
american peopl
dont think
civil war
elect presid
first time
will go
us govern
take place
director jame
jame comey
anthoni weiner
feder govern
middl east
saudi arabia
vote trump
york time
four year
bill clinton
polic offic
year ago
privat email
presid unit
dont know
unit state
presidenti candid
dollar million
secretari state
democrat parti
will never
fbi investig
will make
donald trump
hate
hillari clinton
investig hillari
trump elect
foreign polici
presid obama
presidenti nomine
presidenti elect
democrat parti
everi day
fbi investig
dont know
georg soro
mainstream media
will make
clinton email
human right
unit state
go back
next year
law enforc
take place
secretari state
justic depart
huma abedin
trump will
fox news
post octob
vote hillari
last month
dollar billion
presid unit
can also
york time
washington post
wall street
jame comey
barack obama
republican parti
peopl will
year old
tell us
first time
north carolina
trump said
one can
new york
middl east
anthoni weiner
trump win
just like
mani peopl
two year
social media
million peopl
will never
one day
polic offic
last week
state depart
suprem court
can see
will continu
break news
dollar million
elect day
one thing
help us
fbi director
year later
civil war
white hous
trump campaign
elect presid
look like
will take
support trump
right now
republican presidenti
nation secur
bill clinton
around world
vote trump
last year
year ago
dont want
make sure
unreliable
presid obama
bill clinton
social media
foreign polici
even though
middl east
us militari
john podesta
polic offic
email server
director jame
republican parti
presidenti elect
unit state
human right
us presid
vote trump
vladimir putin
privat email
featur imag
trump victori
first time
fbi director
elect presid
white hous
secretari state
clinton campaign
prime minist
dont want
trump elect
washington dc
fox news
vote hillari
right now
w bush
just like
trump win
anthoni weiner
suprem court
mani peopl
state depart
york time
us govern
last year
dollar million
saudi arabia
dont think
trump said
year old
elect day
share articl
presid unit
dont know
unit nation
world war
jame comey
will go
wall street
make sure
georg w
presidenti campaign
feder govern
us elect
law enforc
reopen investig
washington post
trump campaign
post octob
two year
dollar billion
trump will
berni sander
took place
democrat nation
around world
year ago
new york
one thing
investig hillari
clinton foundat
regim chang
trump support
four year
presidenti candid
nation secur
huma abedin
last week
million peopl
will continu
barack obama
clinton email
mainstream media
civil war
attorney general
obama administr

clickbait



reliable





2.9 Word Associations

Trying to find first associations between terms which are highly correlated:

```r
freq_words <- findFreqTerms(dtms, lowfreq=1000)
for (f in freq_words) {
  print(findAssocs(dtms, f, corlimit=0.8))
}
```

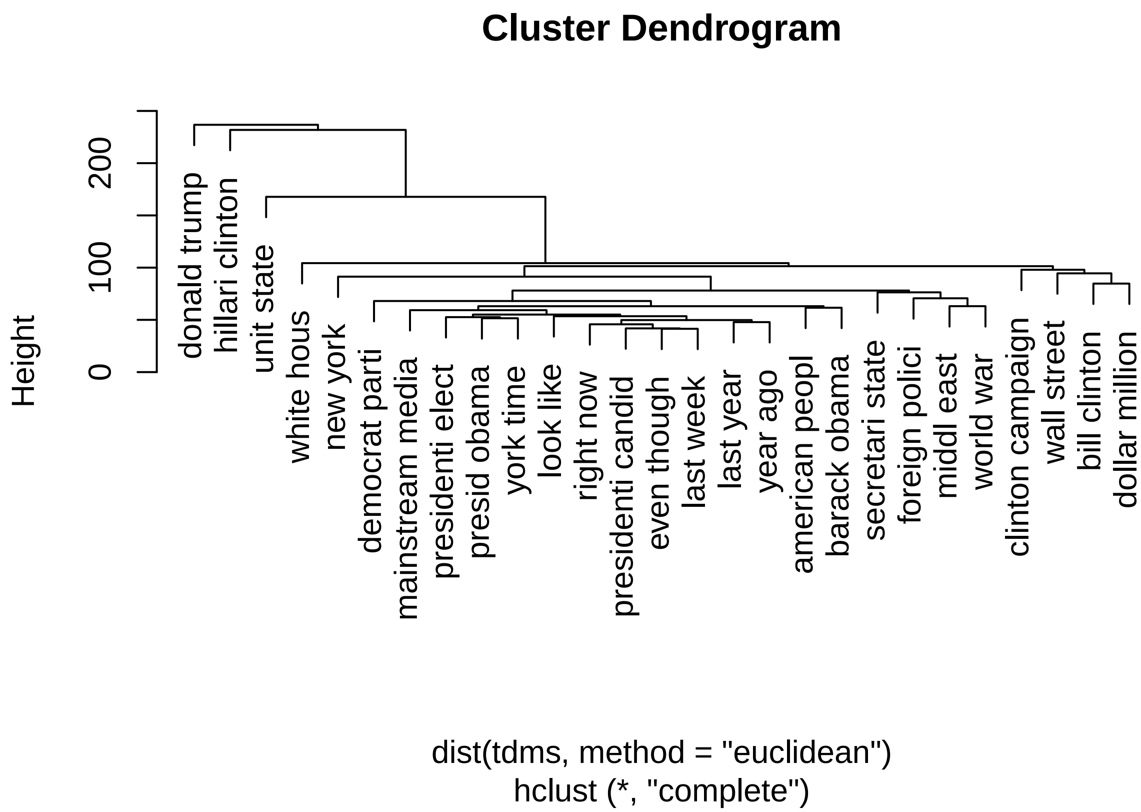
```
## $`american peopl`
## numeric(0)
##
## $`bill clinton`
## numeric(0)
##
## $`clinton campaign`
## numeric(0)
##
## $`clinton foundat`
## numeric(0)
##
## $`dollar million`
## numeric(0)
##
## $`donald trump`
## numeric(0)
##
## $`hillari clinton`
## numeric(0)
##
## $`new york`
## numeric(0)
##
## $`presidenti elect`
## numeric(0)
##
## $`saudi arabia`
## numeric(0)
##
## $`secretari state`
## numeric(0)
##
## $`unit state`
## numeric(0)
##
## $`white hous`
## numeric(0)
```

2.10 Cluster Dendograms

I can also going to have a first deeper look at how the words are hierarchically clustered:

```r
# need to reduce more terms for readibility
tdms <- removeSparseTerms(tdm, 0.955)
# computing euclidean distances between terms
hc <- hclust(d = dist(tdms, method = "euclidean"), method = "complete")
# plotting cluster dendrogram
```

```r
plot(hc)
```



This gives us an interesting overview of the correlations, ergo how specific terms are closer than others to eacher (e.g. Clinton & Wall Street & Dollar / Foreign policy & middle east & world war), but that is at this point of my analysis not helpful. Altough, it kind of indicates which topics/keywords are often mentioned in misleading news together and which are not.

3. Machine Learning

3.1 Split data set

Since our data has been prepared for machine leanring analysis, I now need to split the data into a training dataset and test dataset so that the classifier can be evaluated on data it had not seen previously.

I'll divide the data into two portions: 70 percent for training and 30 percent for testing.

Calculate threshold:

```r
# calculate 30/70 threshold
threshold <- as.integer(round((length(data$text)/100)*30, digit=0))
threshold
```

```
## [1] 3296
```

```r
max_rows <- as.integer(length(data$text))
max_rows
```

```
## [1] 10988
```

(not really necessary, but just in case…) Split raw data:

```
training_raw <- data$text[1:threshold]
testing_raw <- data$text[(threshold+1):max_rows]
```

Split the labels:

```
training_labels <- data$type1[1:threshold]
testing_labels <- data$type1[(threshold+1):max_rows]
```

```
# Comparing the distribution of labels:
training_labels %>% table() %>% prop.table() %>% `*`(100) %>% round(2)
```

```
## .
##       bias  clickbait conspiracy       fake       hate    junksci
##      23.94       7.65      22.24       5.52       6.49       5.89
##  political   reliable      rumor     satire unreliable
##       8.65       0.61       0.88       9.34       8.80
```

```
testing_labels %>% table() %>% prop.table() %>% `*`(100) %>% round(2)
```

```
## .
##       bias  clickbait conspiracy       fake       hate    junksci
##      23.04       6.80      22.79       6.10       6.02       5.84
##  political   reliable      rumor     satire unreliable
##       8.93       0.95       1.01       9.26       9.27
```

The initial shuffling of the dataset was successful: the labels are almost equally distributed among the testing and training dataset.

Splitting the document-term matrix:

```
training_dtm <- dtms[1:threshold,]
testing_dtm <- dtms[(threshold+1):max_rows,]
```

3.2 Creating indicator features for frequent words

Transform the sparse matrix into a data structure that can be used to train a naive Bayes classifier: I will eliminate any words that appear in less than 200 articles in the training data.

Take a document term matrix and returns a character vector containing the words appearing at least a specified number of times.

```
freq_words <- findFreqTerms(training_dtm, 200)
str(freq_words)
```

```
##  chr [1:38] "american peopl" "barack obama" "bill clinton" ...
```

Filter the DTM to include only the terms appearing in the specified vector:

```
training_dtm_freq <- training_dtm[ ,freq_words]
testing_dtm_freq  <- testing_dtm[ ,freq_words]
```

The Naive Bayes classifier is typically trained on data with categorical features. This poses a problem, since the cells in the sparse matrix are numeric and measure the number of times a word appears in a message. We need to change this to a categorical variable that simply indicates yes or no depending on whether the word appears at all.

```
convert_counts <- function(x) {
  x <- ifelse(x > 0, "yes", "no")
```

```r
}

# MARGIN = 1 is used for rows
fake_train <- apply(training_dtm_freq, MARGIN = 2, convert_counts)
fake_test <- apply(testing_dtm_freq, MARGIN = 2, convert_counts)
```

```r
str(fake_train)
```

```
##  chr [1:3296, 1:38] "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" ...
##  - attr(*, "dimnames")=List of 2
##   ..$ Docs : chr [1:3296] "1" "2" "3" "4" ...
##   ..$ Terms: chr [1:38] "american peopl" "barack obama" "bill clinton" "clinton campaign" ...
```

```r
str(fake_test)
```

```
##  chr [1:7692, 1:38] "yes" "no" "no" "no" "no" "no" "no" "no" "no" "no" ...
##  - attr(*, "dimnames")=List of 2
##   ..$ Docs : chr [1:7692] "3297" "3298" "3299" "3300" ...
##   ..$ Terms: chr [1:38] "american peopl" "barack obama" "bill clinton" "clinton campaign" ...
```

3.3 Build training model (with Naive Bayes)

I am going to train my model with the Naive Bayes algorithm which has been proven to be a fast and reliable learning algorithm for text classification:

```r
fake_classifier <- naiveBayes(fake_train, training_labels, laplace=0)
```

3.4 Test model with predictions

```r
t1 = Sys.time()
test_pred <- predict(fake_classifier, fake_test, type="class")
print(difftime(Sys.time(), t1, units = 'sec'))
```

```
## Time difference of 13.7684 secs
```

As we can see, the prediction took less than a minute to calculate.

3.5 Evaluation with Confusion Matrix

```r
# confusion matrix for testing set
confusionMatrix(data=test_pred, testing_labels)
```

```
## Confusion Matrix and Statistics
##
##               Reference
## Prediction   bias clickbait conspiracy fake hate junksci political
##   bias        515       180        438  158  123      69       190
##   clickbait    86        67         80   23    8       3        22
##   conspiracy  370       117        281   74   67      48        93
##   fake         12         5         10    5    0       0         0
##   hate         17         7         13    1    4       4         9
##   junksci      31        13         79   13   22      38        26
##   political    20         3          6    4    1       2         7
```

```
##   reliable     80         9         94    9    9       6        31
##   rumor        11         4         14    1    1       1         2
##   satire      576        99        680  159  223     274       290
##   unreliable   54        19         58   22    5       4        17
##             Reference
## Prediction   reliable rumor satire unreliable
##   bias             12    26     96        217
##   clickbait         2     3     11         47
##   conspiracy        8    17     60        131
##   fake              0     0      0          5
##   hate              0     2      5         13
##   junksci           2     4     14         20
##   political         2     1      1          2
##   reliable         30     2      1         27
##   rumor             0     1      0          2
##   satire           11    20    518        203
##   unreliable        6     2      6         46
##
## Overall Statistics
##
##                Accuracy : 0.1966
##                  95% CI : (0.1877, 0.2056)
##     No Information Rate : 0.2304
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0606
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: bias Class: clickbait Class: conspiracy
## Sensitivity              0.29063          0.12811           0.16030
## Specificity              0.74510          0.96025           0.83415
## Pos Pred Value           0.25445          0.19034           0.22196
## Neg Pred Value           0.77823          0.93787           0.77093
## Prevalence               0.23037          0.06799           0.22790
## Detection Rate           0.06695          0.00871           0.03653
## Detection Prevalence     0.26313          0.04576           0.16459
## Balanced Accuracy        0.51787          0.54418           0.49722
##                      Class: fake Class: hate Class: junksci
## Sensitivity              0.01066    0.008639        0.08463
## Specificity              0.99557    0.990178        0.96907
## Pos Pred Value           0.13514    0.053333        0.14504
## Neg Pred Value           0.93939    0.939740        0.94468
## Prevalence               0.06097    0.060192        0.05837
## Detection Rate           0.00065    0.000520        0.00494
## Detection Prevalence     0.00481    0.009750        0.03406
## Balanced Accuracy        0.50312    0.499409        0.52685
##                      Class: political Class: reliable Class: rumor
## Sensitivity                   0.01019         0.41096      0.01282
## Specificity                   0.99400         0.96482      0.99527
## Pos Pred Value                0.14286         0.10067      0.02703
## Neg Pred Value                0.91103         0.99418      0.98994
## Prevalence                    0.08931         0.00949      0.01014
```

```
## Detection Rate                  0.00091         0.00390      0.00013
## Detection Prevalence            0.00637         0.03874      0.00481
## Balanced Accuracy               0.50210         0.68789      0.50405
##                      Class: satire Class: unreliable
## Sensitivity                  0.72753           0.06452
## Specificity                  0.63682           0.97235
## Pos Pred Value               0.16967           0.19247
## Neg Pred Value               0.95818           0.91051
## Prevalence                   0.09256           0.09269
## Detection Rate               0.06734           0.00598
## Detection Prevalence         0.39691           0.03107
## Balanced Accuracy            0.68217           0.51843
```

Altough the model building and prediction was quickly calculated, the general accuracy is quite low (20%). Even though *accuracy* is probably not the best metric to look at, I will have to find a better way to build my model.

Before we continue with optimising my model in the next steps, I need to mention that 90% of my optimisation took place during the pre-processing phase (cleaning/transforming the text data and number of extracted features).

3.6 Tweak model

3.6.1 LaPlace estimator

A first optimsation can be achieved by enabling the LaPlace estimator. Naive bayes is called "naive" because it does not suppose that the features a correlated and is treating them individually. The Laplace estimator is trying to fix that mathematically.

```
fake_classifier2 <- naiveBayes(fake_train, training_labels, laplace=1)
test_pred2 <- predict(fake_classifier2, fake_test)
cm_bayes11 <- confusionMatrix(data=test_pred2, testing_labels)
```

I am going to save the results of this for later.

3.6.2 Re-classify labels

Another appraoch to tweak the Bayes model is by re-organising my labels. We had a lot of labels and I will now try to group them into three reasonable chunks:

- A: highly misleading
- B: misleading, but not complete fake news
- C: most likely reliable and not misleading content

```
A <- as.factor(c("fake", "conspiracy", "hate", "state"))
B <- as.factor(c("satire", "clickbait", "junksci", "rumor", "bias", "unreliable"))
C <- as.factor(c("reliable", "political"))

training_labels_new <- training_labels
testing_labels_new <- testing_labels

for (r in A) {
  training_labels_new <- gsub(r, 'A', training_labels_new)
  testing_labels_new <- gsub(r, 'A', testing_labels_new)
}
```

```r
for (r in B) {
  training_labels_new <- gsub(r, 'B', training_labels_new)
  testing_labels_new <- gsub(r, 'B', testing_labels_new)
}

for (r in C) {
  training_labels_new <- gsub(r, 'C', training_labels_new)
  testing_labels_new <- gsub(r, 'C', testing_labels_new)
}

training_labels_new = as.factor(training_labels_new)
testing_labels_new = as.factor(testing_labels_new)


training_labels_new %>% table() %>% prop.table() %>% `*`(100) %>% round(2)
```

```
## .
##     A     B     C
## 34.25 56.49  9.25
```

```r
testing_labels_new %>% table() %>% prop.table() %>% `*`(100) %>% round(2)
```

```
## .
##     A     B     C
## 34.91 55.21  9.88
```

```r
fake_classifier3 <- naiveBayes(fake_train, training_labels_new, laplace = 1)
test_pred3 <- predict(fake_classifier3, fake_test)
cm_bayes3 <- confusionMatrix(data=test_pred3, testing_labels_new)
```

Now, I will got a step further and reduce the labels to binary classes:

- A: fake
- B: not fake

```r
# trying FAKE or NOT FAKE
A <- as.factor(c("fake", "conspiracy", "hate", "state", "satire", "junksci",
                 "rumor", "bias", "unreliable", "clickbait"))
B <- as.factor(c("reliable", "political"))

training_labels_new2 <- training_labels
testing_labels_new2 <- testing_labels

for (r in A) {
  training_labels_new2 <- gsub(r, 'A', training_labels_new2)
  testing_labels_new2 <- gsub(r, 'A', testing_labels_new2)
}

for (r in B) {
  training_labels_new2 <- gsub(r, 'B', training_labels_new2)
  testing_labels_new2 <- gsub(r, 'B', testing_labels_new2)
}

training_labels_new2 = as.factor(training_labels_new2)
testing_labels_new2 = as.factor(testing_labels_new2)
```

```r
training_labels_new2 %>% table() %>% prop.table() %>% `*`(100) %>% round(2)
```

```
## .
##     A     B
## 90.75  9.25
```

```r
testing_labels_new2 %>% table() %>% prop.table() %>% `*`(100) %>% round(2)
```

```
## .
##     A     B
## 90.12  9.88
```

```r
fake_classifier4 <- naiveBayes(fake_train, training_labels_new2, laplace = 1)
test_pred4 <- predict(fake_classifier4, fake_test)
cm_bayes2 <- confusionMatrix(data=test_pred4, testing_labels_new2)
```

4. Unsupervised Learning

I am now going to try to build a machine learning model that learns without pre-coded labels. For this approach, I will utilise the K-Means algorithm and run it with 2 K's as in my previous runned model with two classes.

4.1 K-Means

```r
library(stats)
set.seed(23)

#d <- dist(training_dtm, method="euclidian")
kfit_train <- kmeans(training_dtm, 2)

compare_table <- cbind(training_labels_new2, kfit_train$cluster)
compare_table <- as.data.frame(cbind(
                compare_table,ifelse(compare_table[,1]==compare_table[,2],TRUE,FALSE)))
names(compare_table) <- c("label_original","label_kmeans","comparison")

compare_table$comparison %>% table() %>% prop.table() %>% `*`(100) %>% round(2)
```

```
## .
##     0     1
## 17.51 82.49
```

```r
#library(cluster)
#clusplot(as.matrix(d), kfit_train$cluster, color=T, shade=T, labels=2, lines=0)
```

The training phase was pretty fast and a first comparison with the original labels indicates that this unsupervised model achieved from scratch an accuracy over 80%.

But I will now use K-Means to generate three different sets of clusters, run them with Naive Bayes learner and compare them with my previous results.

Comparing Naive Bayes performance with pre-coded labels and different kmeans cluster sizes:

```r
set.seed(23)
```

```r
freq_words <- findFreqTerms(dtms, 200)
dtms_kmeans <- dtms[ ,freq_words]
cm_kmeans_table <- data.frame(matrix(0, ncol = 0, nrow = 7))

#comparing model performance for each label size
for (k in c(11,3,2)) {
  kfit <- kmeans(dtms_kmeans, k)

  k_labels <- as.factor(kfit$cluster)
  k_label_train <- k_labels[1:threshold]
  k_label_test <- k_labels[(threshold+1):max_rows]

  fake_classifier <- naiveBayes(fake_train, k_label_train, laplace = 1)
  test_pred <- predict(fake_classifier, fake_test)
  cm_kmeans <- confusionMatrix(data=test_pred, k_label_test)

  cm_kmeans_table <- cbind(cm_kmeans_table, cm_kmeans$overall)

  if (k==2) {k_labels_2 <- k_labels} # for further analysis
}
```

```r
comparison <- round(100*data.frame(cbind(
                                cm_bayes11$overall,
                                cm_kmeans_table[1],
                                cm_bayes3$overall,
                                cm_kmeans_table[2],
                                cm_bayes2$overall,
                                cm_kmeans_table[3])),2)

names(comparison) <- c("pre-coded11","kmeans11","pre-coded3",
                       "kmeans3", "pre-coded2","kmeans2")
comparison
```

```
##                pre-coded11 kmeans11 pre-coded3 kmeans3 pre-coded2 kmeans2
## Accuracy             19.53    81.47      53.69   90.30      88.66   91.60
## Kappa                 6.09    62.11       3.48   54.05       6.25   55.53
## AccuracyLower        18.65    80.59      52.57   89.62      87.93   90.96
## AccuracyUpper        20.43    82.34      54.81   90.95      89.36   92.21
## AccuracyNull         23.04    69.36      55.21   90.37      90.12   91.52
## AccuracyPValue      100.00     0.00      99.64   58.61     100.00   41.30
## McnemarPValue         0.00      NaN       0.00    0.00       0.00    0.00
```

We can easily see, that the metrics are in general better in regards of K-Means. The accuracy values are highly better than the pre-coded models for 11 and 3 labels - and especially the Kappa coefficient shows more promising results, which is a better metric for imbalanced datasets because it is randomness of occurence taking into consideration.

> Cohen's kappa coefficient (k) is a statistic which measures inter-rater agreement for qualitative (categorical) items. It is generally thought to be a more robust measure than simple percent agreement calculation, as k takes into account the possibility of the agreement occurring by chance.

(Wikipedia 2018a)

Note: Another approach could be the elbow technique, which I tried but removed from this paper,because it didn't show better results.

```r
# further investigation of label/cluster size = 2
compare_table <- as.data.frame(cbind(original_labels, k_labels_2, data$text))
names(compare_table) <- c("label_original","label_kmeans","article")

for (i in c(1,2)) {
  # show corresponding original labels for each cluster that
  # kmeans clustered (sorted by frequency)
  tmp <- data.frame(table(
    compare_table$label_original[compare_table$label_kmeans==i]))
  print(paste("### Cluster", i, "###", sep=" "))
  print(tmp[order(-tmp[2]),])
  # ... and show most frequent bi-grams for each cluster
  tmp2 <- as.vector(compare_table$label_kmeans==i)
  tmp3 <- dtms[,tmp2]
  print(findFreqTerms(tmp3, 500)) # get bi-grams which appear in at least 500 articles
  rm("tmp","tmp2","tmp3")
}
```

```
## [1] "### Cluster 1 ###"
##          Var1 Freq
## 1        bias 2359
## 3  conspiracy 2199
## 10     satire 1010
## 7   political  904
## 11 unreliable  875
## 2   clickbait  689
## 5        hate  654
## 6     junksci  629
## 4        fake  579
## 9       rumor  102
## 8    reliable   59
##  [1] "american peopl"    "anthoni weiner"    "around world"
##  [4] "attorney general"  "barack obama"      "berni sander"
##  [7] "bill clinton"      "clinton campaign"  "clinton email"
## [10] "clinton foundat"   "democrat parti"    "director jame"
## [13] "dollar billion"    "donald trump"      "dont know"
## [16] "elect day"         "email server"      "first time"
## [19] "foreign polici"    "fox news"          "hillari clinton"
## [22] "huma abedin"       "human right"       "jame comey"
## [25] "john podesta"      "justic depart"     "last week"
## [28] "last year"         "law enforc"        "look like"
## [31] "mainstream media"  "mani peopl"        "middl east"
## [34] "nation secur"      "new york"          "presid obama"
## [37] "presidenti candid" "presidenti elect"  "prime minist"
## [40] "right now"         "secretari state"   "social media"
## [43] "state depart"      "suprem court"      "trump campaign"
## [46] "trump support"     "us govern"         "us militari"
## [49] "vladimir putin"    "wall street"       "washington post"
## [52] "will continu"      "world war"         "year ago"
## [55] "york time"
## [1] "### Cluster 2 ###"
##          Var1 Freq
## 3  conspiracy  287
## 1        bias  202
```

```
## 11 unreliable  128
## 2   clickbait   86
## 4        fake   72
## 7   political   68
## 8    reliable   34
## 5        hate   23
## 6     junksci   14
## 10     satire   10
## 9       rumor    5
## [1] "dollar million"  "even though"     "fbi director"    "obama administr"
## [5] "saudi arabia"    "unit state"      "white hous"
```

Promimently can be seen that especially the tags "bias" and "conspiracy" are in both cluster frequently represented, even though the size of the second cluster is largely smaller by 90 percent. The top topics in the second cluster are related to the (by now former) FBI director Comey, the Obama administration and political and economical ties to Saudi Arabia. The first cluster contains all the keywords during the presidential duel which were aimed against the candidate democratic candidate Hillary Clinton: the Anthony Weiner scandal, her close associate Huma Abedin (ex-wife of Weiner), John Podesta and the e-mail server scandal, the Clinton Foundation, her ties to Wall Street and general terms against the "mainstream media". It would be quite interesting to dive deeper into the two categories, bias and conspiracy, because they are both not clearly to differ from "conventional, established news" due to their usage of the same terminology of "reliable" language and terms for the common newspaper reader. But these kind of articles are usually easy to detect how they refer to sources, how many reliable sources refer to them and how strong their arguments are, because biased and conspiracy-ish articles are typically mixing known facts and connecting non-related dots with each other to "create" new "truths" and insights.

5. Smote

Due to the initially recognised problem that my dataset is highly imbalanced (only 10% are labelled as "not fake") I will need to apply a combined sampling method called "SMOTE", which is over-sampling the minority class ("not fake") and under-sampling the majority class ("fake").

(Chawla et al. 2002)

```r
library(DMwR)
set.seed(23)
df <- data.frame(cbind(as.character(data$text),as.character(data$type1)))
names(df) <- c("text","label")
```

```r
df$label <- ifelse(df$label == "reliable","NOT_FAKE",
                   ifelse(df$label=="political","NOT_FAKE","FAKE"))
df$label <- as.factor(df$label)
round(100*prop.table(table(df$label)),2)
```

```
##
##     FAKE NOT_FAKE
##    90.31     9.69
```

```r
balanced <- SMOTE(label ~ ., df, perc.over = 300, perc.under = 200)
round(100*prop.table(table(balanced$label)),2)
```

```
##
##     FAKE NOT_FAKE
##       60       40
```

```
set.seed(23)
balanced2 <- balanced[sample(nrow(balanced)),]
balanced <- balanced2

threshold <- as.integer(round((length(balanced$text)/100)*30, digit=0))
max_rows <- as.integer(length(balanced$text))
training_labels <- balanced$label[1:threshold]
testing_labels <- balanced$label[(threshold+1):max_rows]

round(100*prop.table(table(training_labels)),0)
```

```
## training_labels
##     FAKE NOT_FAKE
##       60       40
```

```
round(100*prop.table(table(testing_labels)),0)
```

```
## testing_labels
##     FAKE NOT_FAKE
##       60       40
```

Run everything again with balanced dataset:

```
t1 = Sys.time()

# text pre-processing
## preparing corpus
balanced$text <- gsub(balanced$text, pattern = "’", replacement = "'")
balanced$text <- gsub(balanced$text, pattern = "-", replacement = " ")
pre_corpus <- bracketX(balanced$text)
pre_corpus <- replace_abbreviation(pre_corpus)
pre_corpus <- replace_symbol(pre_corpus)
## creating and removing noise from corpus
dfCorpus = VCorpus(VectorSource(pre_corpus))
dfCorpus <- tm_map(dfCorpus, content_transformer(tolower))
dfCorpus <- tm_map(dfCorpus, removeNumbers)
dfCorpus <- tm_map(dfCorpus, removePunctuation)
dfCorpus <- tm_map(dfCorpus, content_transformer(function(x)
  gsub(x,pattern = "'re",replacement = " are")))
dfCorpus <- tm_map(dfCorpus, content_transformer(function(x)
  gsub(x,pattern = "'m",replacement = " am")))
dfCorpus <- tm_map(dfCorpus, content_transformer(function(x)
  gsub(x,pattern = "'ve",replacement = " have")))
dfCorpus <- tm_map(dfCorpus, content_transformer(function(x)
  gsub(x,pattern = "n't",replacement = " not")))
replace_list <- c("/", "@", "\\|", "\"", """, """, "-", "-", "‘", "_", "'s")
for (r in replace_list) {
  dfCorpus <- tm_map(dfCorpus, content_transformer(function(x)
    gsub(x,pattern =  r, replacement = "  ")))
}
dfCorpus <- tm_map(dfCorpus, removeWords,
                   c(stopwords("english"),
                   "http","https","www","dot","com","pictwitt"))
dfCorpus <- tm_map(dfCorpus, stripWhitespace)
dfCorpus <- tm_map(dfCorpus, stemDocument)
```

```r
# creating document term matrix and bi-gram tokenization
bigram_tokenizer <- function(x) {
  RWeka::NGramTokenizer(x, RWeka::Weka_control(min=2, max=2))}
dtm <- DocumentTermMatrix(dfCorpus,control=list(tokenize = bigram_tokenizer))
dtms <- removeSparseTerms(dtm, 0.98)
# reducing and splitting dataset/DTM
training_dtm <- dtms[1:threshold,]
testing_dtm <- dtms[(threshold+1):max_rows,]
freq_words <- findFreqTerms(training_dtm, 200)
training_dtm_freq <- training_dtm[ ,freq_words]
testing_dtm_freq  <- testing_dtm[ ,freq_words]
## normalise test and train data (convert frequencies to binary factors)
convert_counts <- function(x) { x <- ifelse(x > 0, "yes", "no") }
fake_train <- apply(training_dtm_freq, MARGIN = 2, convert_counts)
fake_test <- apply(testing_dtm_freq, MARGIN = 2, convert_counts)
# train model and evaluating prediction performance
## naive bayes with balanced dataset
fake_classifier <- naiveBayes(fake_train, training_labels, laplace = 1)
test_pred <- predict(fake_classifier, fake_test, type="class")
cm_balanced <- confusionMatrix(data=test_pred, testing_labels)
## k-means with balanced dataset
freq_words <- findFreqTerms(dtms, 200)
dtms_kmeans <- dtms[ ,freq_words]
kfit <- kmeans(dtms_kmeans, k)
k_labels <- as.factor(kfit$cluster)
k_label_train <- k_labels[1:threshold]
k_label_test <- k_labels[(threshold+1):max_rows]
fake_classifier <- naiveBayes(fake_train, k_label_train, laplace = 1)
test_pred <- predict(fake_classifier, fake_test)
cm_balanced_kmeans <- confusionMatrix(data=test_pred, k_label_test)

print(difftime(Sys.time(), t1, units = 'min'))
```

```
## Time difference of 15.93914 mins
```

... with Cross-Validation:

```r
set.seed(23)

cv_train <- data.frame(cbind(fake_train, k_label_train))
cv_test <- data.frame(cbind(fake_test, k_label_test))

train_control <- trainControl(method="cv", number=10)
fake_classifier <- train(k_label_train ~., data=cv_train,
                         trControl=train_control, method="nb")
test_pred <- predict(fake_classifier, cv_test, type="raw")
cm_balanced_cv <- confusionMatrix(data=test_pred, cv_test$k_label_test)
```

Comparing all metrics:

```r
#comparison of all performances for balanced dataset
smote_comparison <- round(100*data.frame(cbind(
                      cm_balanced$overall,
                      cm_balanced_cv$overall,
                      cm_balanced_kmeans$overall)),2)
```

```r
names(smote_comparison) <- c("smote_normal", "smote_CV", "smote_kmeans")
smote_comparison
```

```
##                 smote_normal smote_CV smote_kmeans
## Accuracy               72.14    96.94       100.00
## Kappa                  34.94     0.00       100.00
## AccuracyLower          71.11    96.53        99.95
## AccuracyUpper          73.16    97.32       100.00
## AccuracyNull           59.99    96.94        96.94
## AccuracyPValue          0.00    51.76         0.00
## McnemarPValue           0.00     0.00          NaN
```

6. Conclusions

At first, I should emphasize that this text classification project was based on a dataset with a restricted, not very diverse, heterogene topic (US presidential campaign), the timeframe of data collection was limited (one month) and the dataset is vastly imbalanced (10% non-misleading sources). Furthermore, I didn't have included the websites, the authors, text length, social media data and further meta data into my analyses.

But this investigation can be start to build better and more generic algorithms for automatic fake news classification.

We also saw, that this dataset required a huge amount of time and code for pre-processing, and actually also requires a deeper qualitative investigation, in order to disinguish and assess news articles.

Altough, the two-label approach (fake or not fake) showed good performance results, the further K-Means clustering analysis indicates that certain labels are not easily distinguishable and/or one label per atricle is too superficial.

To be honest, I am slightly confused about the last performance metrics regarding the *SMOTE balanced* dataset. The normal NB based classification perfomed weaker in terms of Kappa and Accuracy than with the imbalanced data, the 10-fold cross validation resulted in the highest level of accuracy so far, but a Kappa value of 0 implies, that the accuracy was just achieved by picking picking only the rows with the highest frequency and calculating the means for each class, but the algorithm was not robust enough after a ten fold validation. Regarding the *smote_kmeans* results: the values of 100% look very suspicious and are probably a sign of over-ftting, which should be checked in further calculations, e.g. with a test, train and *validation* dataset.

In summary, the unsupervised modelling to identify first clusters for different text categories and then digging deeper qualitatively to identify more suitable clusters/labels, and finally execute and compare several fast supervised models (NB, SVM etc.) seems to be the better, more stable approach. Also, a comprehensive literature review of academic research papers would be more than beneficial towrds this matter. But nevertheless, this investigation has presented more than interesting insights into the "hot topic" fake news.

References

Chawla, Nitesh V, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. "SMOTE: Synthetic Minority over-Sampling Technique." *Journal of Artificial Intelligence Research* 16: 321–57.

(https://stats.stackexchange.com/users/86949/indi), Indi. n.d. *What Is VectorSource and VCorpus in Text Mining Package in R.* https://stats.stackexchange.com/q/169467.

Wikipedia. 2018a. *Cohen's Kappa - Wikipedia.* http://en.wikipedia.org/w/index.php?title=Cohen's%

20kappa&oldid=837232308.

———. 2018b. *N-Gram - Wikipedia.* http://en.wikipedia.org/w/index.php?title=N-gram&oldid=835900923.