Tweet Sentiment Analysis

Dr. Jeffrey Strickland

9/13/2018

Sentiment Lexicons

One way to analyze the sentiment of a text is to consider the text as a combination of its individual words and the sentiment content of the whole text as the sum of the sentiment content of the individual words. This is often performed using lexicons. Linguistic theories generally regard human languages as consisting of two parts: a lexicon, essentially a catalogue of a language's words (its wordstock); and a grammar, a system of rules which allow for the combination of those words into meaningful sentences. Lexicons used of sentiment analysis contain sentiment words, like "trust" and "disgust," and all the words associated with a particular sentiment. These lexicons are "lined" togther with the text under analysis by performing an inner-join with the text, the details of which we will discuss later.

Sentiment lexicons constructed via either crowdsourcing (using, for example, Amazon Mechanical Turk) or by the labor of one of the authors, and are validated using some combination of crowdsourcing again, restaurant or movie reviews, or Twitter data. Given this information, we may hesitate to apply these sentiment lexicons to styles of text dramatically different from what they were validated on, such as narrative fiction from 200 years ago. While it is true that using these sentiment lexicons with "Indian Philosophy," for example, may give us less accurate results than with tweets sent by a contemporary writer, we still can measure the sentiment content for words that are shared across the lexicon and the text.

The three general-purpose lexicons are - AFINN from Finn Årup Nielsen, - bing from Bing Liu and collaborators, and - nrc from Saif Mohammad and Peter Turney.

All three of these lexicons are based on unigrams, i.e., single words. These lexicons contain many English words and the words are assigned scores for positive/negative sentiment, and also possibly emotions like joy, anger, sadness, and so forth. The nrc lexicon categorizes words in a binary fashion ("yes"/"no") into categories of positive, negative, anger, anticipation, disgust, fear, joy, sadness, surprise, and trust. The bing lexicon categorizes words in a binary fashion into positive and negative categories. The AFINN lexicon assigns words with a score that runs between -5 and 5, with negative scores indicating negative sentiment and positive scores indicating positive sentiment. All of this information is tabulated in the sentiments dataset, and tidytext provides a function get_sentiments() to get specific sentiment lexicons without the columns that are not used in that lexicon.

Not every English word is in the lexicons because many English words are pretty neutral. It is important to keep in mind that these methods do not take into account qualifiers before a word, such as in “no good” or “not true”; a lexicon-based method like this is based on unigrams only.

Install Required Libraries

```
    if(!require(tidytext)) install.packages("tidytext")

## Loading required package: tidytext

    if(!require(tidyverse)) install.packages("tidyverse")

## Loading required package: tidyverse

## -- Attaching packages ---------------------------------------------------
-------- tidyverse 1.2.1 --

## v ggplot2 3.0.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.6
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0

## -- Conflicts ------------------------------------------------------------
-- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

    if(!require(dplyr)) install.packages("dplyr")
    if(!require(tidyr)) install.packages("tidyr")
    if(!require(sentimentr)) install.packages("sentimentr")

## Loading required package: sentimentr

    if(!require(tm)) install.packages("tm")

## Loading required package: tm

## Loading required package: NLP

##
## Attaching package: 'NLP'

## The following object is masked from 'package:ggplot2':
##
##     annotate

    if(!require(readr)) install.packages("readr")
    if(!require(wordcloud)) install.packages("wordcloud")

## Loading required package: wordcloud

## Loading required package: RColorBrewer
```

```r
    if(!require(lubridate)) install.packages("lubridate")

## Loading required package: lubridate

##
## Attaching package: 'lubridate'

## The following object is masked from 'package:base':
##
##     date

    if(!require(ggplot2)) install.packages("ggplot2")
    if(!require(ggraph)) install.packages("ggraph")

## Loading required package: ggraph

    if(!require(igraph)) install.packages("igraph")

## Loading required package: igraph

##
## Attaching package: 'igraph'

## The following objects are masked from 'package:lubridate':
##
##     %--%, union

## The following objects are masked from 'package:dplyr':
##
##     as_data_frame, groups, union

## The following objects are masked from 'package:purrr':
##
##     compose, simplify

## The following object is masked from 'package:tidyr':
##
##     crossing

## The following object is masked from 'package:tibble':
##
##     as_data_frame

## The following objects are masked from 'package:stats':
##
##     decompose, spectrum

## The following object is masked from 'package:base':
##
##     union

    if(!require(plotrix)) install.packages("plotrix")

## Loading required package: plotrix
```

Lexicon Example

The tidytext package contains several sentiment lexicons in the sentiments dataset.

```
library(tidytext)
sentiment
```

```
## function (text.var, polarity_dt = lexicon::hash_sentiment_jockers_rinker,
##     valence_shifters_dt = lexicon::hash_valence_shifters, hyphen = "",
##     amplifier.weight = 0.8, n.before = 5, n.after = 2, question.weight =
1,
##     adversative.weight = 0.85, neutral.nonverb.like = FALSE,
##     missing_value = 0, ...)
## {
##     UseMethod("sentiment")
## }
## <bytecode: 0x000000002113b470>
## <environment: namespace:sentimentr>
```

We can look at the way specific lexicons score sentiment. The bing lexicon categorizes words in a binary fashion into positive and negative categories.

```
get_sentiments("bing")
```

```
## # A tibble: 6,788 x 2
##    word        sentiment
##    <chr>       <chr>
##  1 2-faced     negative
##  2 2-faces     negative
##  3 a+          positive
##  4 abnormal    negative
##  5 abolish     negative
##  6 abominable  negative
##  7 abominably  negative
##  8 abominate   negative
##  9 abomination negative
## 10 abort       negative
## # ... with 6,778 more rows
```

The AFINN lexicon assigns words with a score that runs between -5 and 5, with negative scores indicating negative sentiment and positive scores indicating positive sentiment.

```
get_sentiments("afinn")
```

```
## # A tibble: 2,476 x 2
##    word       score
##    <chr>      <int>
##  1 abandon       -2
##  2 abandoned     -2
##  3 abandons      -2
##  4 abducted      -2
##  5 abduction     -2
##  6 abductions    -2
##  7 abhor         -3
##  8 abhorred      -3
##  9 abhorrent     -3
## 10 abhors        -3
## # ... with 2,466 more rows
```

The nrc lexicon categorizes words in a binary fashion (“yes”/“no”) into categories of positive, negative, anger, anticipation, disgust, fear, joy, sadness, surprise, and trust.

```
get_sentiments("nrc")
```

```
## # A tibble: 13,901 x 2
##    word        sentiment
##    <chr>       <chr>
##  1 abacus      trust
##  2 abandon     fear
##  3 abandon     negative
##  4 abandon     sadness
##  5 abandoned   anger
##  6 abandoned   fear
##  7 abandoned   negative
##  8 abandoned   sadness
##  9 abandonment anger
## 10 abandonment fear
## # ... with 13,891 more rows
```

Example 1: The Inner Join

With data in a tidy format, sentiment analysis can be done as an inner-join. This is another of the great successes of viewing text mining as a tidy data analysis task; much as removing stop words is an anti-join operation, performing sentiment analysis is an inner-join operation. For this example, we will use a collection of tweets from President Donald Trump, as the lexicons are geared more toward the analysis of tweets than the analysis of ancient philosophies.

```
setwd("C:/Users/jeff/Documents/VIT_Course_Material/Data_Analytics_2018/data/"
)
trump_tweets<-read.csv("trump_tweets.csv",stringsAsFactor=FALSE)
str(trump_tweets)
```

```
## 'data.frame':    1050 obs. of  2 variables:
##  $ time  : chr  "8/13/2018 8:57" "8/13/2018 9:21" "8/11/2018 1:28"
"8/8/2018 10:14" ...
##  $ tweets: chr  " ....such wonderful and powerful things about me - a true
Champion of Civil Rights - until she got fired. Omaro"| __truncated__ " While
I know it�s �not presidential� to take on a lowlife like Omarosa, and while I
would rather not be doing "| __truncated__ " The big story that the Fake News
```

```
Media refuses to report is lowlife Christopher Steele�s many meetings with
De"| __truncated__ " The Republicans have now won 8 out of 9 House Seats, yet
if you listen to the Fake News Media you would think "| __truncated__ ...
```

Observe that trump_tweets is alerady a data frame, and that there are 1050 tweets in the dataset, with line one as a header. We will look at the words with a "trust" score from the NRC lexicon. What are the most common "trust" words in Trump_Tweets? First, we need to take the text of the novels and convert the text to the tidy format using unnest_tokens().

The next code chunk converts the time to date in the dataset, which will be ordered by date.

```
trump_tweets$date<-as.Date(trump_tweets$time, "%m/%d/%Y %H:%M")
head(trump_tweets)

##              time
## 1 8/13/2018 8:57
## 2 8/13/2018 9:21
## 3 8/11/2018 1:28
## 4 8/8/2018 10:14
## 5  8/5/2018 7:49
## 6  8/5/2018 7:35
##
tweets
## 1  ....such wonderful and powerful things about me - a true Champion of
Civil Rights - until she got fired. Omarosa had Zero credibility with the
Media (they didn�t want interviews) when she worked in the White House. Now
that she says bad about me, they will talk to her. Fake News! [Twitter for
iPhone] link
## 2       While I know it�s �not presidential� to take on a lowlife like
Omarosa, and while I would rather not be doing so, this is a modern day form
of communication and I know the Fake News Media will be working overtime to
make even Wacky Omarosa look legitimate as possible. Sorry! [Twitter for
iPhone] link
## 3  The big story that the Fake News Media refuses to report is lowlife
Christopher Steele�s many meetings with Deputy A.G. Bruce Ohr and his
beautiful wife, Nelly. It was Fusion GPS that hired Steele to write the phony
& discredited Dossier, paid for by Crooked Hillary & the DNC.... [Twitter for
iPhone] link
## 4                                      The Republicans have now won 8 out
of 9 House Seats, yet if you listen to the Fake News Media you would think we
are being clobbered. Why can�t they play it straight, so unfair to the
Republican Party and in particular, your favorite President! [Twitter for
iPhone] link
## 5
Too bad a large portion of the Media refuses to report the lies and
corruption having to do with the Rigged Witch Hunt - but that is why we call
them FAKE NEWS! [Twitter for iPhone] link
## 6    Fake News reporting, a complete fabrication, that I am concerned
about the meeting my wonderful son, Donald, had in Trump Tower. This was a
meeting to get information on an opponent, totally legal and done all the
```

```
time in politics - and it went nowhere. I did not know about it! [Twitter for
iPhone] link
##          date
## 1 2018-08-13
## 2 2018-08-13
## 3 2018-08-11
## 4 2018-08-08
## 5 2018-08-05
## 6 2018-08-05
```

Next, we will tokenize the text.

```
trump_tweets<-trump_tweets %>% group_by(date) %>% mutate(ln=row_number())%>%
unnest_tokens(word,tweets) %>% ungroup()
head(trump_tweets,5)

## # A tibble: 5 x 4
##   time           date          ln word
##   <chr>          <date>     <int> <chr>
## 1 8/13/2018 8:57 2018-08-13     1 such
## 2 8/13/2018 8:57 2018-08-13     1 wonderful
## 3 8/13/2018 8:57 2018-08-13     1 and
## 4 8/13/2018 8:57 2018-08-13     1 powerful
## 5 8/13/2018 8:57 2018-08-13     1 things
```

Notice that we chose the name word for the output column from unnest_tokens(). This makes performing inner joins and anti-joins is thus easier because the sentiment lexicons and stop word datasets have columns named word.

Now that the text is in a tidy format with one word per row, we are ready to do the sentiment analysis. First, let's use the NRC lexicon and filter() for the "trust" words. Next, we will filter() the data frame with the text from the trump_tweets for the words and then use inner_join() to perform the sentiment analysis. What are the most common "trust" words in trump_tweets? We'll use count() from dplyr get this answer.

```
nrc_trust <- get_sentiments("nrc") %>%
  filter(sentiment == "trust")
trump_tweets %>%
  inner_join(nrc_trust) %>%
  count(word, sort = TRUE)

## Joining, by = "word"

## # A tibble: 189 x 2
##    word          n
##    <chr>     <int>
##  1 president    63
##  2 show         31
##  3 enjoy        25
##  4 good         20
##  5 vote         20
##  6 real          19
##  7 credibility   18
##  8 money         18
##  9 trade         17
## 10 white         17
## # ... with 179 more rows
```

We see mostly positive words associated with "trust". Now we'll look an see what "disgust" the President has.

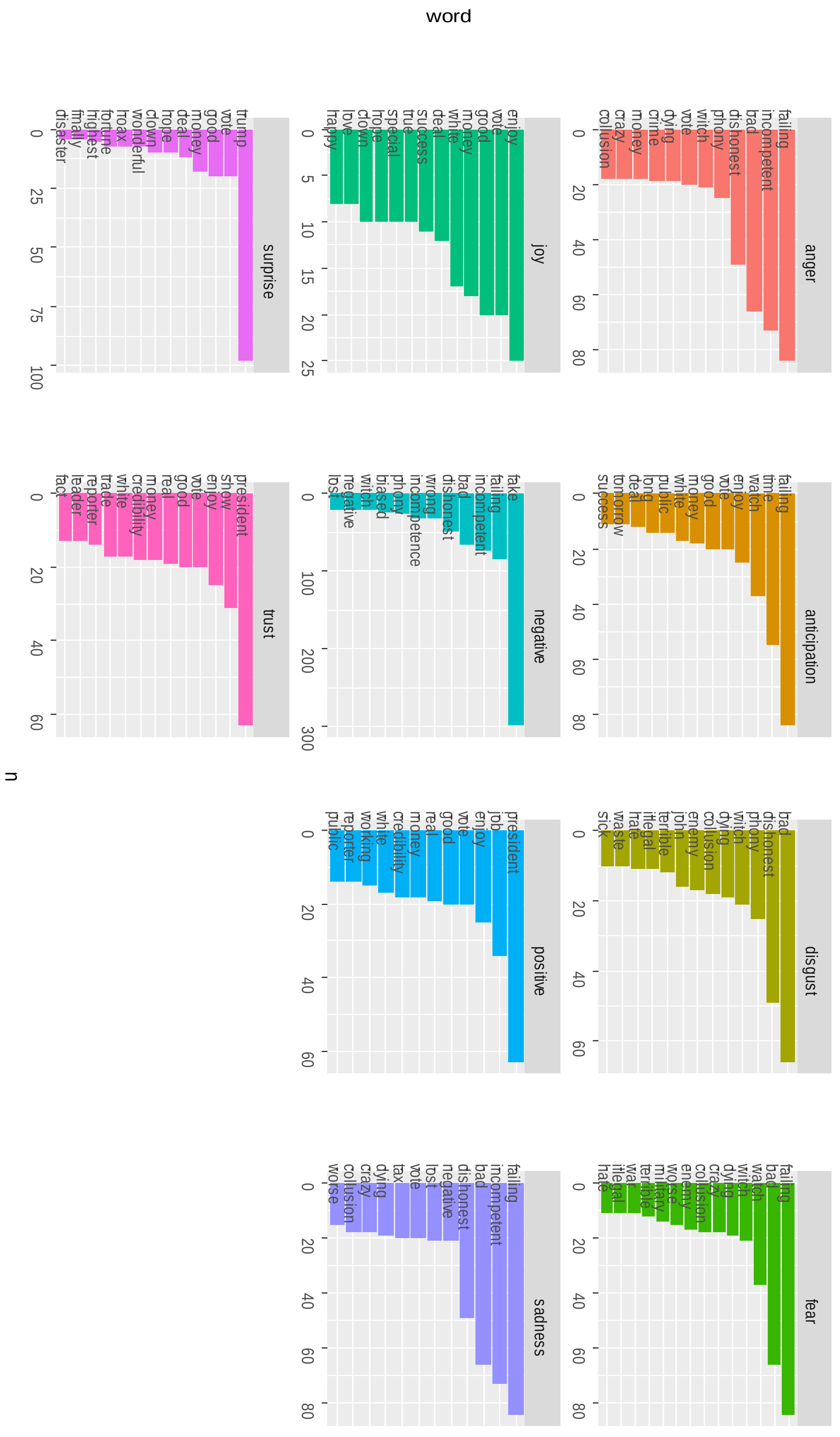
```
nrc_disgust <- get_sentiments("nrc") %>%
  filter(sentiment == "disgust")
trump_tweets %>%
  inner_join(nrc_disgust) %>%
  count(word, sort = TRUE)

## Joining, by = "word"

## # A tibble: 126 x 2
##    word          n
##    <chr>     <int>
##  1 bad          66
##  2 dishonest    49
##  3 phony        25
##  4 witch        21
##  5 dying        19
##  6 collusion    18
##  7 enemy        17
##  8 john         16
##  9 terrible     12
## 10 hate         11
## # ... with 116 more rows
```

We can also examine how sentiment changes throughout the time period (10/19/2011 to 8/13/2018). We can do this with just a handful of lines that are mostly dplyr functions. First, we find a sentiment score for each word using the Bing lexicon and inner_join(). Next, we count up how many positive and negative words there are in defined in each tweet. We then use spread() so that we have negative and positive sentiment in separate columns, and lastly calculate a net sentiment (positive - negative).

```
library(tidyr)
trump_sentiment <- trump_tweets %>% inner_join(get_sentiments("bing"))

## Joining, by = "word"
```

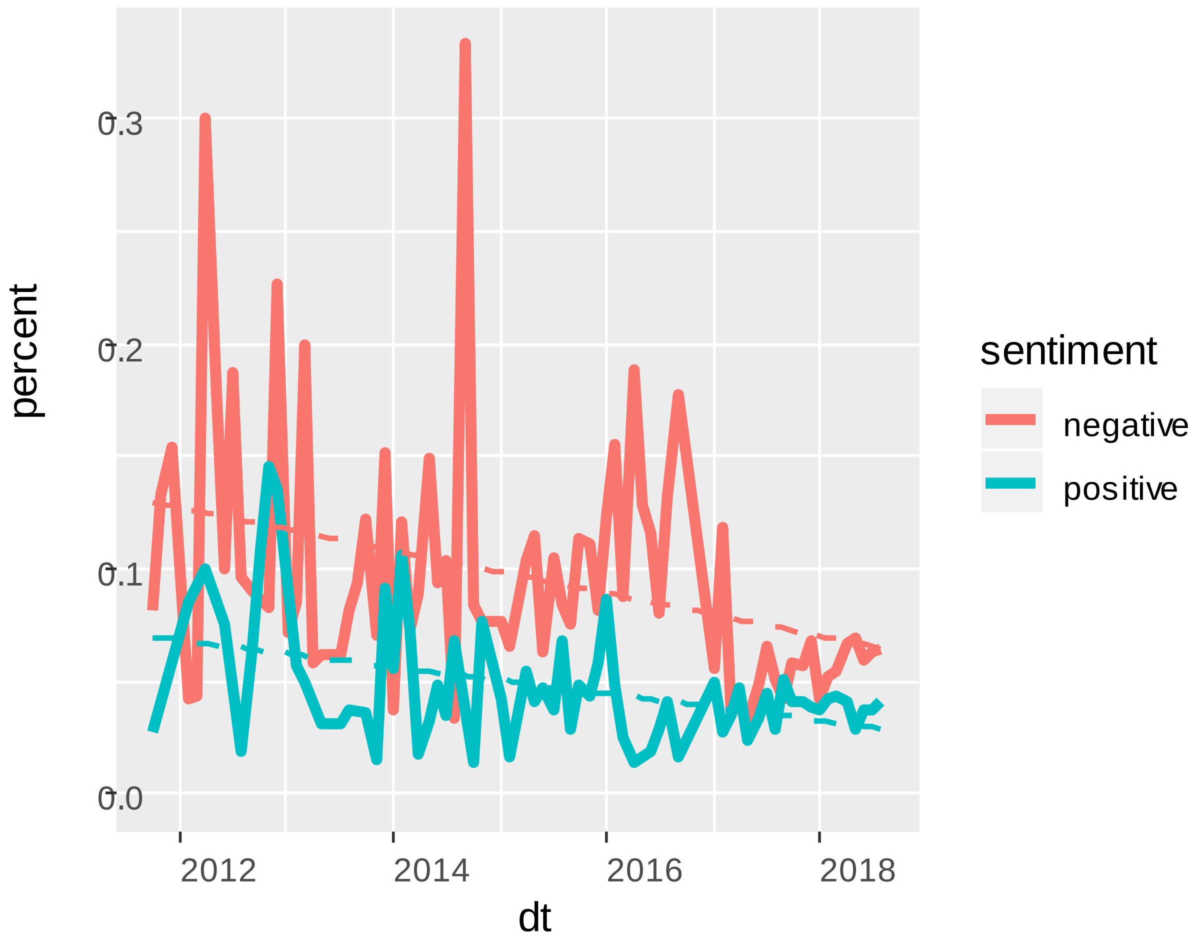
Notice that we are plotting against the index on the x-axis that keeps track of narrative time in sections of text.

```
library(ggplot2)
trump_tweets%>%inner_join(get_sentiments("nrc")) %>%count(word,sentiment) %>%
group_by(sentiment)%>%top_n(10)%>%
  ungroup()
%>%mutate(word=reorder(word,n))%>%ggplot(aes(x=word,y=n,fill=sentiment)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ sentiment, scales = "free") +
  coord_flip()

## Joining, by = "word"

## Selecting by n
```

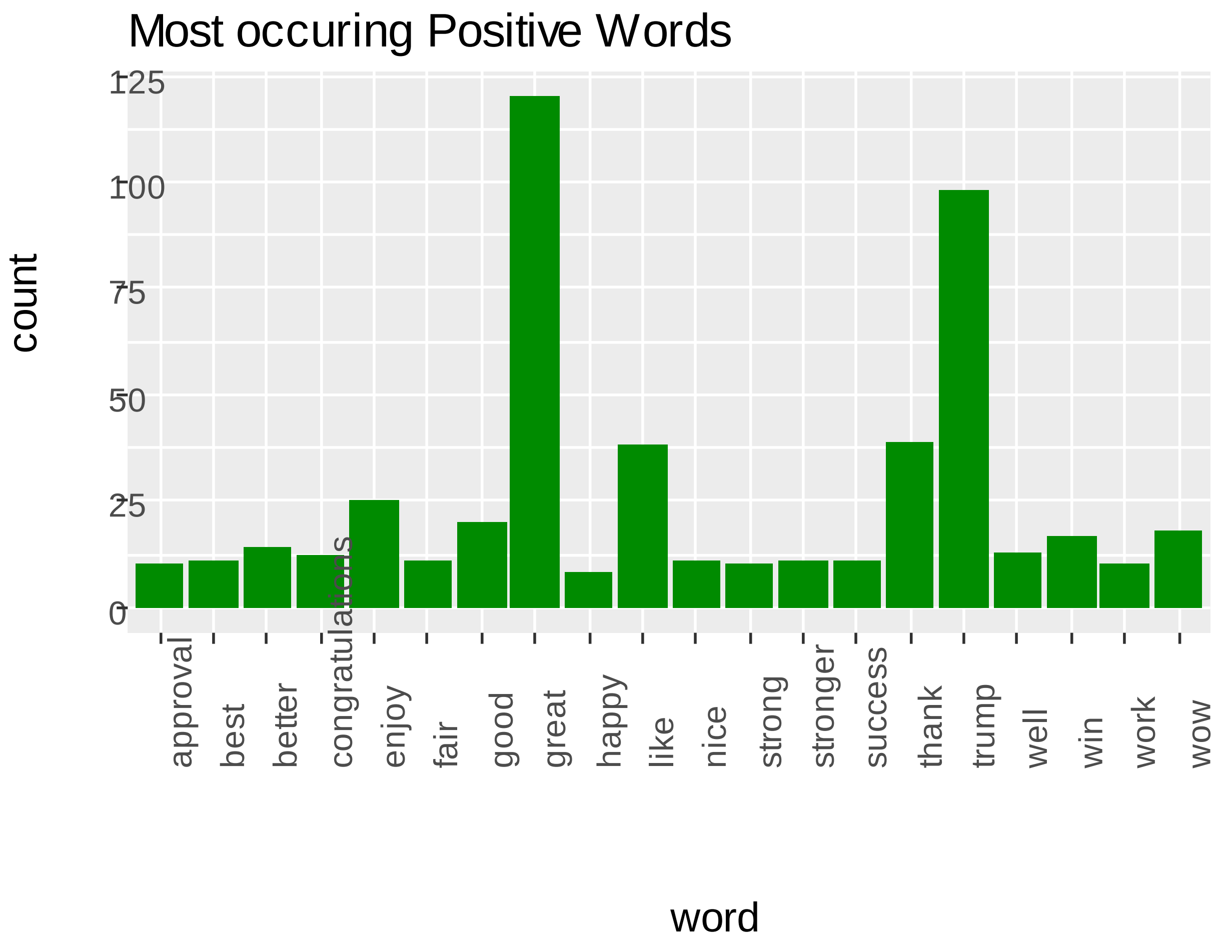

word
n
anger
failing
incompetent
bad
dishonest
phony
witch
vote
dying
crime
money
crazy
collusion
anticipation
failing
time
watch
enjoy
vote
good
money
white
public
long
deal
tomorrow
success
disgust
bad
dishonest
phony
witch
dying
collusion
enemy
john
terrible
illegal
hate
waste
sick
fear
failing
bad
watch
witch
dying
crazy
collusion
enemy
worse
military
terrible
war
illegal
hate
joy
enjoy
vote
good
money
white
deal
success
true
special
hope
clown
love
happy
negative
fake
failing
incompetent
bad
dishonest
wrong
incompetence
phony
biased
witch
negative
lost
positive
president
job
enjoy
vote
good
real
money
credibility
white
working
reporter
public
sadness
failing
incompetent
bad
dishonest
negative
lost
vote
tax
dying
crazy
collusion
worse
surprise
trump
vote
good
money
deal
hope
clown
wonderful
hoax
fortune
highest
finally
disaster
trust
president
show
enjoy
vote
good
real
money
credibility
white
trade
reporter
leader
fact

Positive & Negative Words over time

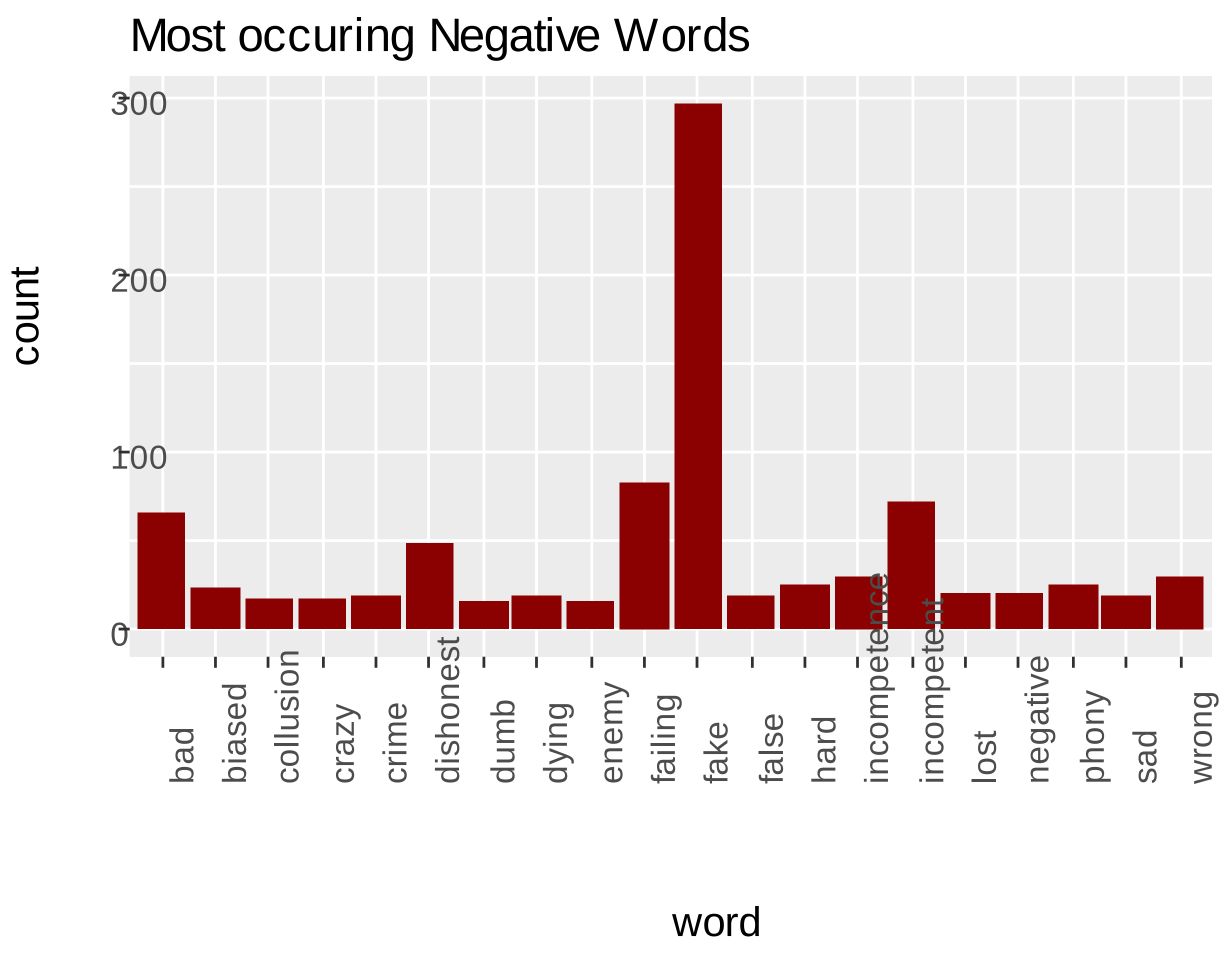
```
sentiment_by_time <- trump_tweets %>%
  mutate(dt = floor_date(date, unit = "month")) %>%
  group_by(dt) %>%
  mutate(total_words = n()) %>%
  ungroup() %>%
  inner_join(get_sentiments("nrc"))

## Joining, by = "word"

sentiment_by_time %>%
  filter(sentiment %in% c('positive','negative')) %>%
  count(dt,sentiment,total_words) %>%
  ungroup() %>%
  mutate(percent = n / total_words) %>%
  ggplot(aes(x=dt,y=percent,col=sentiment,group=sentiment)) +
  geom_line(size = 1.5) +
  geom_smooth(method = "lm", se = FALSE, lty = 2) +
  expand_limits(y = 0)
```



```
pos_neg<-trump_sentiment %>%count(word,sentiment,sort=TRUE)
pos_neg %>% filter(sentiment=='positive')%>%head(20)
%>%ggplot(aes(x=word,y=n))+geom_bar(stat="identity",fill="green4")+
  theme(axis.text.x=element_text(angle=90))+labs(title="Most occuring
Positive Words",y="count")
```



```
pos_neg %>% filter(sentiment=='negative')%>%head(20)
%>%ggplot(aes(x=word,y=n))+geom_bar(stat="identity",fill="red4")+
  theme(axis.text.x=element_text(angle=90))+labs(title="Most occuring
Negative Words",y="count")
```



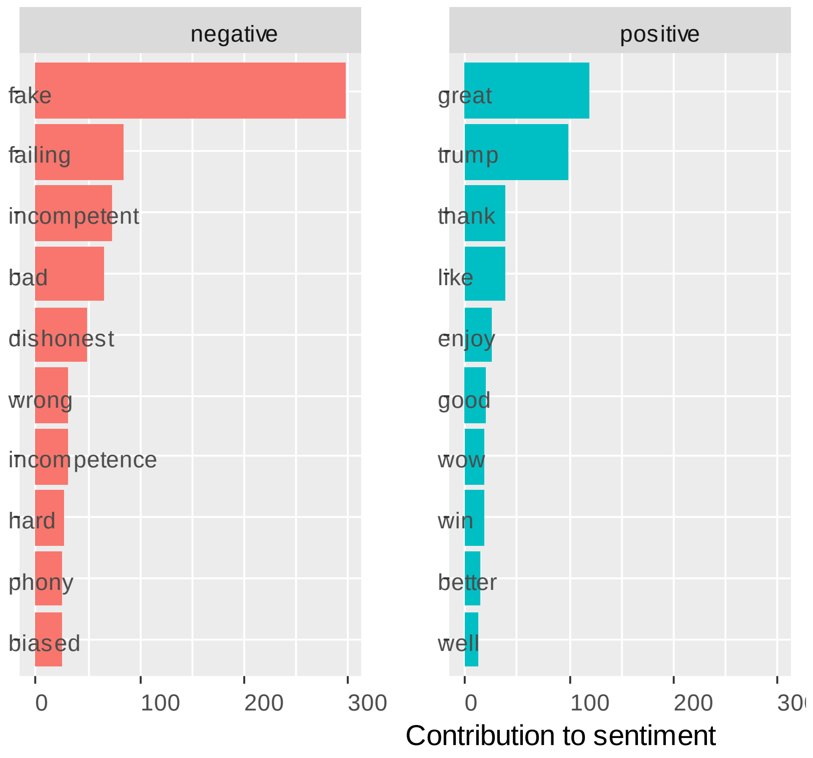
Most common positive and negative words

One advantage of having the data frame with both sentiment and word is that we can analyze word counts that contribute to each sentiment. By implementing count() here with arguments of both word and sentiment, we find out how much each word contributed to each sentiment.

```
bing_word_counts <- trump_tweets %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  ungroup()

## Joining, by = "word"

bing_word_counts

## # A tibble: 543 x 3
##    word          sentiment     n
##    <chr>         <chr>     <int>
##  1 fake          negative    298
##  2 great         positive    120
##   3 trump        positive     98
##   4 failing      negative     84
##   5 incompetent  negative     73
##   6 bad          negative     66
##   7 dishonest    negative     49
##   8 thank        positive     39
##   9 like         positive     38
##  10 incompetence negative     31
## # ... with 533 more rows
```

This can be shown visually, and we can pipe straight into ggplot2, if we like, because of the way we are consistently using tools built for handling tidy data frames.

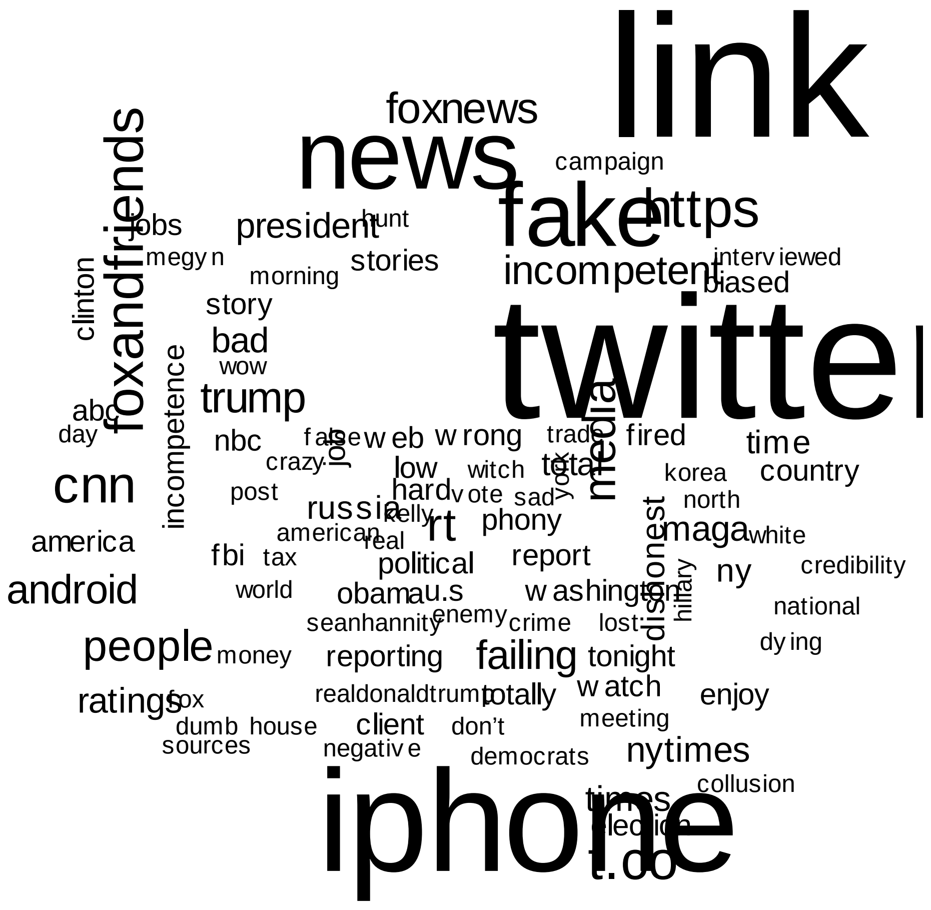
```
bing_word_counts %>%
  group_by(sentiment) %>%
  top_n(10) %>%
  ungroup() %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n, fill = sentiment)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~sentiment, scales = "free_y") +
  labs(y = "Contribution to sentiment",
       x = NULL) +
  coord_flip()

## Selecting by n
```



Wordclouds

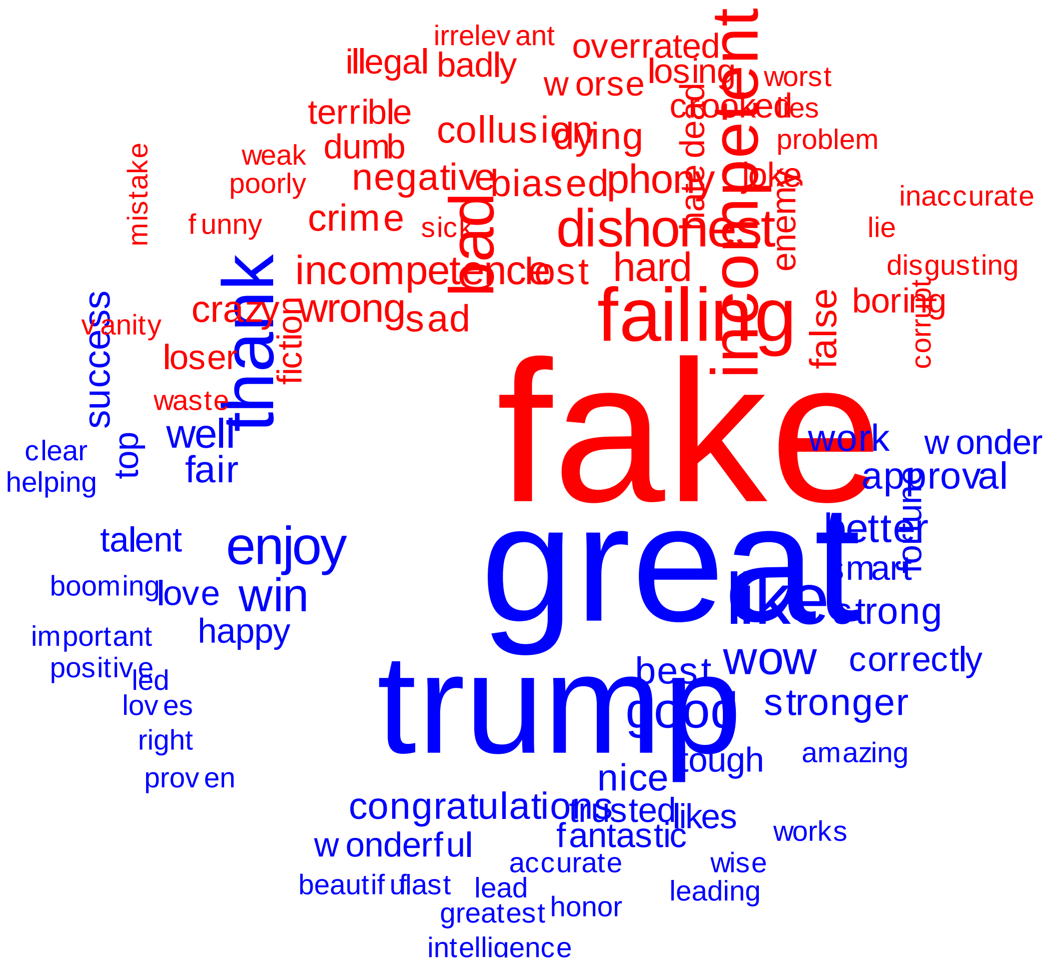
We've seen that this tidy text mining approach works well with ggplot2, but having our data in a tidy format is useful for other plots as well. For example, consider the wordcloud package, which uses base R graphics. Let's look at the most common words in trump_tweets, but this time as a wordcloud.

```r
library(wordcloud)
trump_tweets %>%
  anti_join(stop_words) %>%
  count(word) %>%
  with(wordcloud(word, n, max.words = 100))
```



In other functions, such as comparison.cloud(), you may need to turn the data frame into a matrix with reshape2's acast(). Let's do the sentiment analysis to tag positive and negative words using an inner join, then find the most common positive and negative words. Until the step where we need to send the data to comparison.cloud(), this can all be done with joins, piping, and dplyr because our data is in tidy format.

```
library(reshape2)

##
## Attaching package: 'reshape2'

## The following object is masked from 'package:tidyr':
##
##     smiths

trump_tweets %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(colors = c("red", "blue"),
                   max.words = 100)

## Joining, by = "word"
```



Word Frequencies

In this next section, we want to look at trump_tweets from the perspective of Mr. Trump's different roles, assuming that might be some sentiment differences between Candidate Trump and President Trump. The column "role" contains a flag to delineate Mr. Trump's different roles. After January 21, 2017, his role is "president" and prior to that, his role is "candidate."

The next code chunk removes user-made and standard English stopwords.

```
library(tm)
library(tidyr)
my_stops <- c("https", "a", "rt", "t.co", "for", "they", stopwords("en"))
trump_tweets <- trump_tweets %>%
  filter(!word %in% stop_words$word,
         !word %in% my_stops,
         !word %in% str_remove_all(stop_words$word, "'"))
```

Now we can calculate word frequencies for each Donald Trump role. First, we group by person and count how many times each role used each word. Then we use left_join() to add a column of the total number of words used by each role.

```
frequency <- trump_tweets %>%
  group_by(role) %>%
  count(word, sort = TRUE) %>%
  left_join(trump_tweets %>%
              group_by(role) %>%
              summarise(total = n())) %>%
  mutate(freq = n/total)

## Joining, by = "role"

frequency

## # A tibble: 3,719 x 5
## # Groups:   role [2]
##    role      word              n total    freq
##    <chr>     <chr>         <int> <int>   <dbl>
##  1 president twitter         584  9947 0.0587
##  2 president link            575  9947 0.0578
##  3 president iphone          541  9947 0.0544
##  4 president news            320  9947 0.0322
##  5 president fake            298  9947 0.0300
##  6 president foxandfriends   143  9947 0.0144
##  7 president media           109  9947 0.0110
##  8 president cnn             106  9947 0.0107
##  9 candidate link             91  3178 0.0286
## 10 president foxnews          89  9947 0.00895
## # ... with 3,709 more rows
```

This is a nice and tidy data frame but we would actually like to plot those frequencies on the x- and y-axes of a plot, so we will need to use spread() from tidyr make a differently shaped data frame.

```r
if(!require(tidyr)) install.packages("tidyr")
library(tidyr)
frequency <- frequency %>%
  select(role, word, freq) %>%
  spread(role, freq) %>%
  arrange(president, candidate)
frequency
```

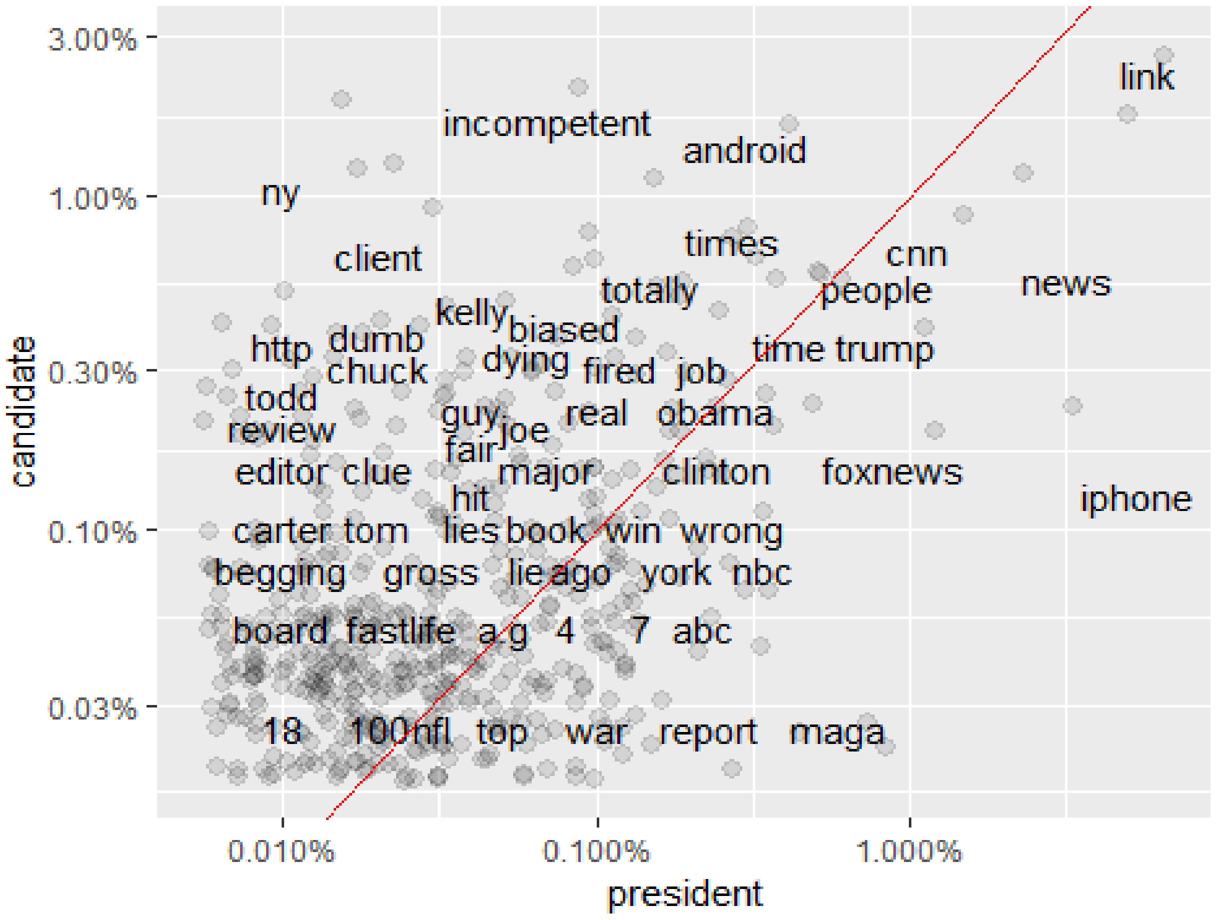
```
## # A tibble: 3,210 x 3
##    word      candidate president
##    <chr>         <dbl>     <dbl>
##  1 18         0.000315  0.000101
##  2 admitted   0.000315  0.000101
##  3 agreed     0.000315  0.000101
##  4 allowing   0.000315  0.000101
##  5 articles   0.000315  0.000101
##  6 attempt    0.000315  0.000101
##  7 average    0.000315  0.000101
##  8 basic      0.000315  0.000101
##  9 bob        0.000315  0.000101
## 10 buy        0.000315  0.000101
## # ... with 3,200 more rows
```

Now this is ready for us to plot. We will use geom_jitter() so that we don't see the discreteness at the low end of frequency as much, and check_overlap = TRUE so the text labels don't all print out on top of each other (only some will print).

```r
if(!require(scales)) install.packages("scales")
```

```
## Loading required package: scales

##
## Attaching package: 'scales'

## The following object is masked from 'package:plotrix':
##
##     rescale

## The following object is masked from 'package:purrr':
##
##     discard

## The following object is masked from 'package:readr':
##
##     col_factor
```

```r
library(scales)
ggplot(frequency, aes(president, candidate)) +
  geom_jitter(alpha = 0.1, size = 2.5, width = 0.25, height = 0.25) +
  geom_text(aes(label = word), check_overlap = TRUE, vjust = 1.5) +
  scale_x_log10(labels = percent_format()) +
  scale_y_log10(labels = percent_format()) +
  geom_abline(color = "red")

## Warning: Removed 2701 rows containing missing values (geom_point).

## Warning: Removed 2701 rows containing missing values (geom_text).
```



Words near the line are used with about equal frequencies by President Trump and Candidate Trump, while words far away from the line are used much more by one person compared to the other. Words, hashtags, and usernames that appear in this plot are ones that we have both used at least once in tweets.

Comparing word usage

We just made a plot comparing raw word frequencies over our whole Twitter histories; now let’s find which words are more or less likely to come from each Mr. Trump’s roles, using the log odds ratio.

Here, we count how many times each role uses each word and keep only the words used more than 10 times. After a spread() operation, we can calculate the log odds ratio for each word, using

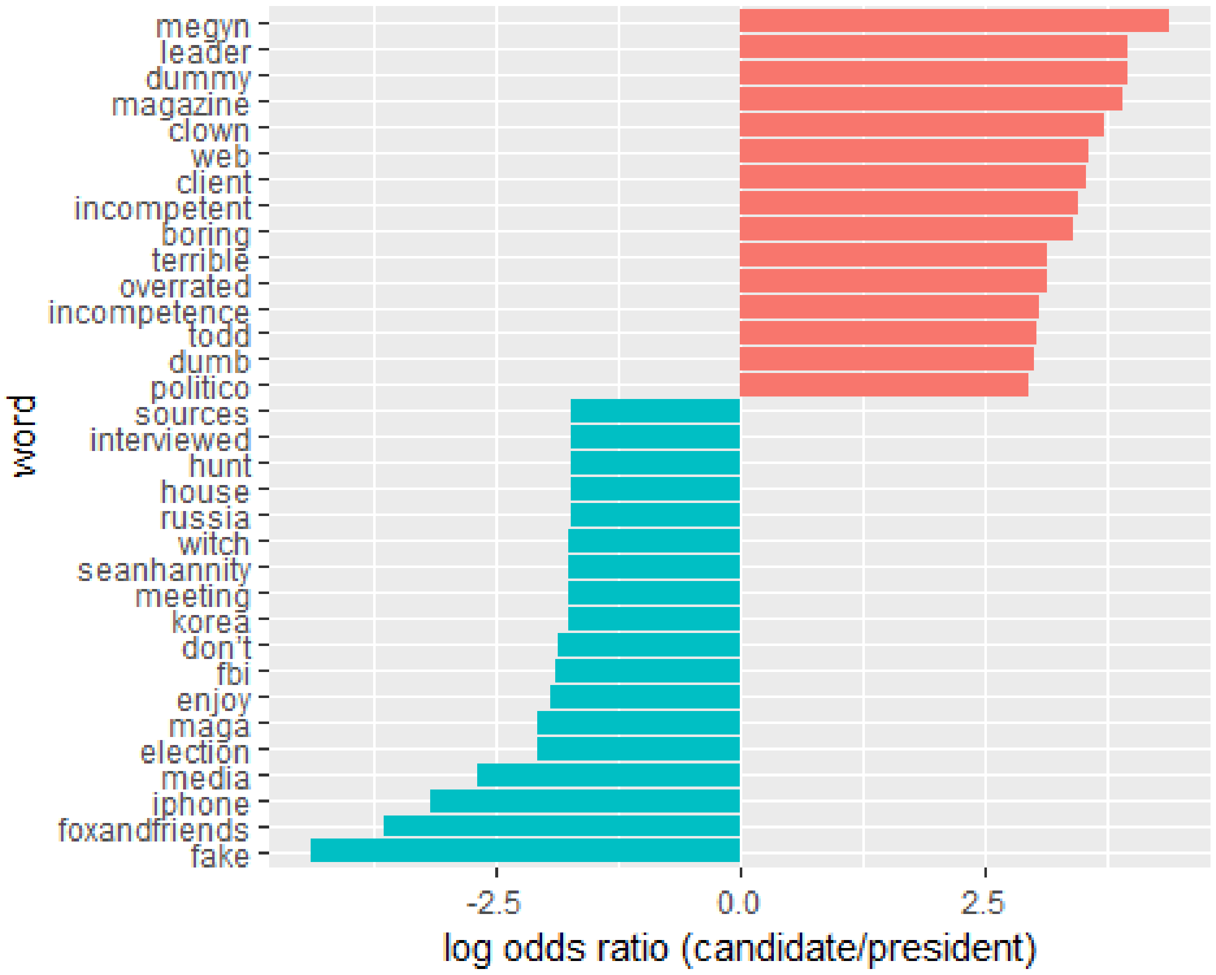
log odd ratio = ln(((n+1/(tital+1))president)/(((n+1)/(total+1))candidate))

where n is the number of times the word in question is used by each role and the total indicates the total words for each role.

```
my_stops <- c("https", "http", "a", "rt", "t.co", "for", "they", "north",
"ny", stopwords("en"))
trump_tweets <- trump_tweets %>%
  filter(!word %in% stop_words$word,
         !word %in% my_stops,
         !word %in% str_remove_all(stop_words$word, "'"))
word_ratios <- trump_tweets %>%
  filter(!str_detect(word, "^@")) %>%
  count(word, role) %>%
  group_by(word) %>%
  filter(sum(n) >= 10) %>%
  ungroup() %>%
  spread(role, n, fill = 0) %>%
  mutate_if(is.numeric, funs((. + 1) / (sum(.) + 1))) %>%
  mutate(logratio = log(candidate / president)) %>%
  arrange(desc(logratio))
word_ratios %>%
  arrange(abs(logratio))

## # A tibble: 203 x 4
##    word     candidate president logratio
##    <chr>        <dbl>     <dbl>    <dbl>
##  1 cnn        0.0191    0.0192  -0.00422
##  2 people     0.0150    0.0145   0.0330
##  3 polls      0.00205   0.00197  0.0371
##  4 china      0.00205   0.00215 -0.0499
##  5 coverage   0.00205   0.00215 -0.0499
##  6 million    0.00205   0.00215 -0.0499
##  7 campaign   0.00273   0.00287 -0.0499
##  8 world      0.00273   0.00287 -0.0499
##  9 clinton    0.00478   0.00449  0.0634
## 10 u.s        0.00478   0.00521 -0.0850
## # ... with 193 more rows
```

What are some words that have been about equally likely to come from Candidate Trump or President Trump?

```
word_ratios %>%
  group_by(logratio < 0) %>%
  top_n(15, abs(logratio)) %>%
  ungroup() %>%
  mutate(word = reorder(word, logratio)) %>%
  ggplot(aes(word, logratio, fill = logratio < 0)) +
  geom_col(show.legend = FALSE) +
  coord_flip() +
  ylab("log odds ratio (candidate/president)") +
  scale_fill_discrete(name = "", labels = c("candidate", "president"))
```



Changes in word use

The section above looked at overall word use, but now let's ask a different question. Which words' frequencies have changed the fastest in Mr.Trumps Twitter feeds? Or to state this another way, which words has he tweeted about at a higher or lower rate as time has passed in his different roles? To do this, we will define a new time variable in the data frame that defines which unit of time each tweet was posted in. We can use floor_date() from lubridate to do this, with a unit of our choosing; using 1 month seems to work well for set of tweets.

After we have the time bins defined, we count how many times each of us used each word in each time bin. After that, we add columns to the data frame for the total number of words used in each time bin by each role and the total number of times each word was used by each role We can then filter() to only keep words used at least some minimum number of times (30, in this case).

```
words_by_time <- trump_tweets %>%
  filter(!str_detect(word, "^@")) %>%
  mutate(time_floor = floor_date(date, unit = "1 month")) %>%
  count(time_floor, role, word) %>%
  group_by(role, time_floor) %>%
  mutate(time_total = sum(n)) %>%
  group_by(role, word) %>%
  mutate(word_total = sum(n)) %>%
  ungroup() %>%
  rename(count = n) %>%
  filter(word_total > 30)
words_by_time

## # A tibble: 482 x 6
##    time_floor role      word        count time_total word_total
##    <date>     <chr>     <chr>       <int>      <int>      <int>
##  1 2011-10-01 candidate incompetent     2         24         66
##  2 2011-10-01 candidate link            2         24         91
##  3 2011-11-01 candidate incompetent     1          9         66
##  4 2011-11-01 candidate link            1          9         91
##  5 2012-06-01 candidate link            1         20         91
##  6 2012-06-01 candidate twitter         1         20         86
##  7 2012-08-01 candidate incompetent     1         28         66
##  8 2012-08-01 candidate link            1         28         91
##  9 2012-09-01 candidate incompetent     4         49         66
## 10 2012-09-01 candidate link            4         49         91
## # ... with 472 more rows
```

Each row in this data frame corresponds to one role using one word in a given time bin. The count column tells us how many times that role used that word in that time bin, the time_total column tells us how many words that role used during that time bin, and the word_total column tells us how many times that person used that word over the whole year. This is the data set we can use for modeling.

We can use nest() from tidyr to make a data frame with a list column that contains little miniature data frames for each word. Let's do that now and take a look at the resulting structure.

```
nested_data <- words_by_time %>%
  nest(-word, -role)
nested_data

## # A tibble: 25 x 3
##    role      word        data
##    <chr>     <chr>       <list>
##  1 candidate incompetent <tibble [31 x 4]>
##  2 candidate link        <tibble [39 x 4]>
##   3 candidate twitter     <tibble [36 x 4]>
##   4 candidate android     <tibble [29 x 4]>
##   5 president cnn         <tibble [18 x 4]>
##   6 president failing     <tibble [15 x 4]>
##   7 president fake        <tibble [19 x 4]>
##   8 president foxnews     <tibble [19 x 4]>
##   9 president iphone      <tibble [19 x 4]>
##  10 president link        <tibble [20 x 4]>
## # ... with 15 more rows
```

This data frame has one row for each role-word combination; the data column is a list column that contains data frames, one for each combination of role and word. Let's use map() from purrr (Henry and Wickham 2018) to apply our modeling procedure to each of those little data frames inside our big data frame. This is count data so let's use glm() with family = "binomial" for modeling.

```
if(!require(purrr)) install.packages("purrr")
library(purrr)
nested_models <- nested_data %>%
  mutate(models = map(data, ~ glm(cbind(count, time_total) ~ time_floor, .,
                                  family = "binomial")))
nested_models
```

```
## # A tibble: 25 x 4
##    role      word        data              models
##    <chr>     <chr>       <list>            <list>
##  1 candidate incompetent <tibble [31 x 4]> <S3: glm>
##  2 candidate link        <tibble [39 x 4]> <S3: glm>
##  3 candidate twitter     <tibble [36 x 4]> <S3: glm>
##  4 candidate android     <tibble [29 x 4]> <S3: glm>
##  5 president cnn         <tibble [18 x 4]> <S3: glm>
##  6 president failing     <tibble [15 x 4]> <S3: glm>
##  7 president fake        <tibble [19 x 4]> <S3: glm>
##  8 president foxnews     <tibble [19 x 4]> <S3: glm>
##  9 president iphone      <tibble [19 x 4]> <S3: glm>
## 10 president link        <tibble [20 x 4]> <S3: glm>
## # ... with 15 more rows
```

Now notice that we have a new column for the modeling results; it is another list column and contains glm objects. The next step is to use map() and tidy() from the broom package to pull out the slopes for each of these models and find the important ones. We are comparing many slopes here and some of them are not statistically significant, so let's apply an adjustment to the p-values for multiple comparisons.

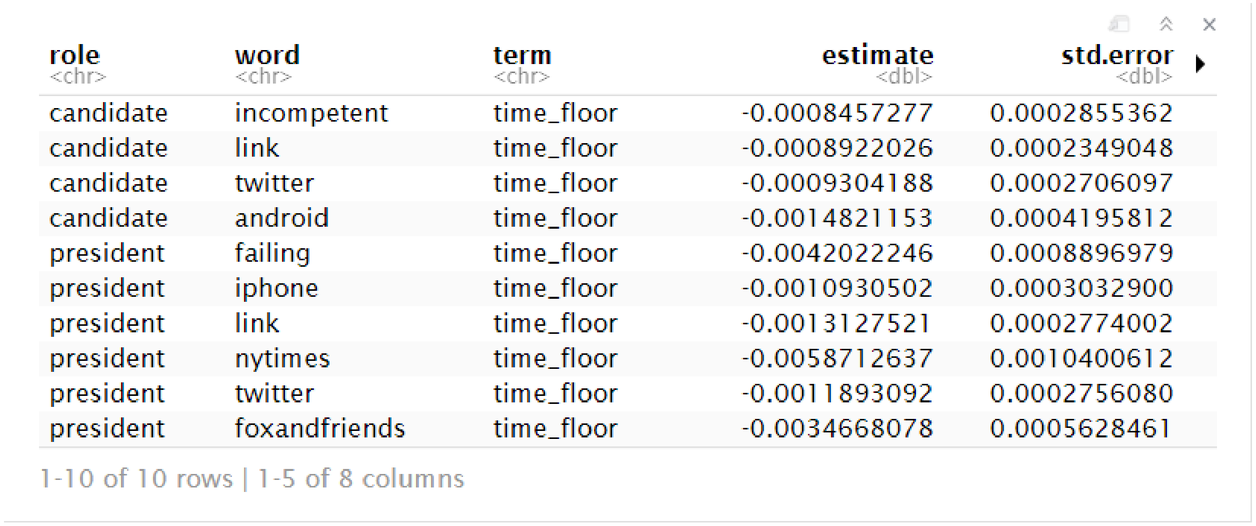
```
if(!require(broom)) install.packages("broom")
```

```
## Loading required package: broom
```

```
library(broom)
library(tidyr)
slopes <- nested_models %>%
  unnest(map(models, tidy)) %>%
  filter(term == "time_floor") %>%
  mutate(adjusted.p.value = p.adjust(p.value))
```

Now let's find the most important slopes. Which words have changed in frequency at a moderately significant level in our tweets?

```
top_slopes <- slopes %>%

  filter(adjusted.p.value < 0.05)

top_slopes
```

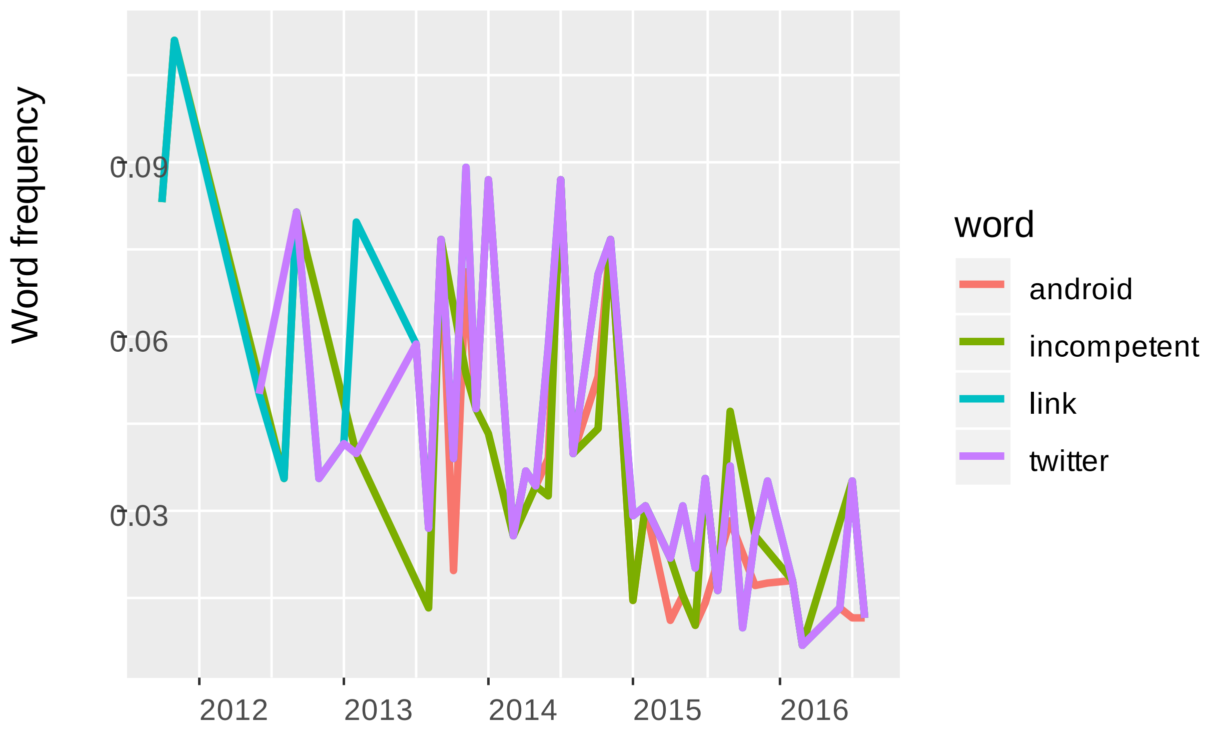


role <chr>	word <chr>	term <chr>	estimate <dbl>	std.error <dbl>
candidate	incompetent	time_floor	-0.0008457277	0.0002855362
candidate	link	time_floor	-0.0008922026	0.0002349048
candidate	twitter	time_floor	-0.0009304188	0.0002706097
candidate	android	time_floor	-0.0014821153	0.0004195812
president	failing	time_floor	-0.0042022246	0.0008896979
president	iphone	time_floor	-0.0010930502	0.0003032900
president	link	time_floor	-0.0013127521	0.0002774002
president	nytimes	time_floor	-0.0058712637	0.0010400612
president	twitter	time_floor	-0.0011893092	0.0002756080
president	foxandfriends	time_floor	-0.0034668078	0.0005628461

1-10 of 10 rows | 1-5 of 8 columns

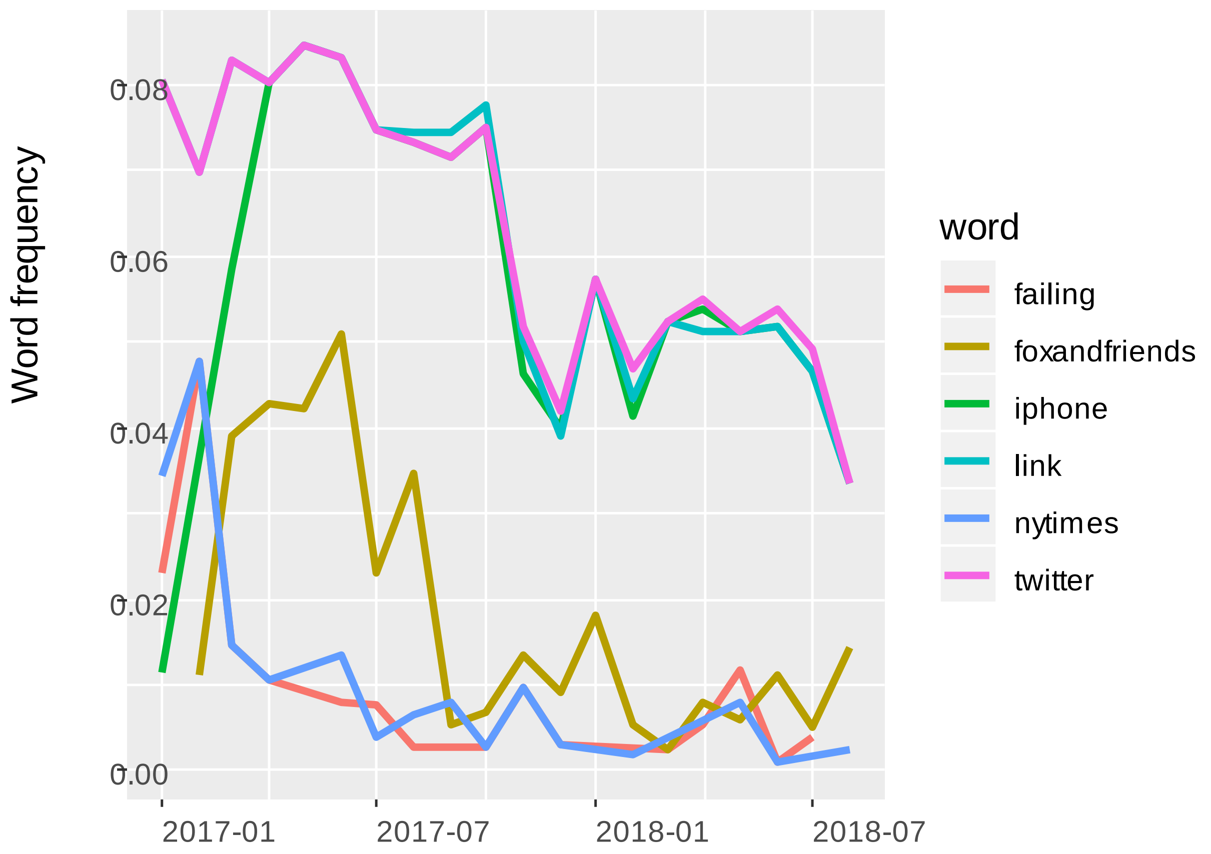
To visualize our results, we can plot these words' use for both Candidate Trump and President Trump over tweet-period.

```
words_by_time %>%

  inner_join(top_slopes, by = c("word", "role")) %>%

  filter(role == "candidate") %>%

  ggplot(aes(time_floor, count/time_total, color = word)) +

  geom_line(size = 1.3) +

  labs(x = NULL, y = "Word frequency")
```



Now let’s plot words that have changed frequency in President Trump's tweets.

```
words_by_time %>%
  inner_join(top_slopes, by = c("word", "role")) %>%
  filter(role == "president") %>%
  ggplot(aes(time_floor, count/time_total, color = word)) +
  geom_line(size = 1.3) +
  labs(x = NULL, y = "Word frequency")
```



Example 2: IsisFanboy

Sourcing Data

```
setwd("C:/Users/jeff/Documents/VIT_University/IsisFanboy/")
getwd()

## [1] "C:/Users/jeff/Documents/VIT_University/IsisFanboy"

tweets<-read.csv("AboutIsis.csv",stringsAsFactor=FALSE)
str(tweets)

## 'data.frame':    122619 obs. of  5 variables:
##  $ name    : chr  "Sean Ferigan" "m.zakariyya" "ã\201¡ã��ã��ã��ã\201�"
"chutney" ...
##  $ username: chr  "ferigan" "mzakariyya5" "yuzuchaaan777" "plainparatha"
...
##  $ tweetid : num  7.52e+17 7.52e+17 7.52e+17 7.52e+17 7.52e+17 ...
##  $ time    : chr  "7/11/2016 8:45:39 AM" "7/11/2016 8:45:39 AM" "7/11/2016
8:45:38 AM" "7/11/2016 8:45:38 AM" ...
##  $ tweets  : chr  "ISIS influence on the decline as terrorists lose
Twitter battles     - CNET http://www.cnet.com/news/isis-influ"|
__truncated__ "RT @AyishaBaloch: #IndiaISISandBangladesh And world can ALSO
```

```
not ignore the truth revealing india 's role in pr"| __truncated__
"@Laika_isis @wink_BF
ã©ãªã¢ã\201£ã\201¦ã\201ªã\201ã\201ã\201ã\201" "RT @KabirTaneja:
Anti-ISIS volunteer fighting with the Kurds. things are getting strange on
planet Earth.  #Pok"| __truncated__ ...
```

Processing Data

View of data

```
tweets$date<-as.Date(tweets$time, "%d/%m/%Y %H:%M:%S")
head(tweets)

##                    name      username      tweetid                  time
## 1          Sean Ferigan       ferigan 7.524236e+17 7/11/2016 8:45:39 AM
## 2           m.zakariyya   mzakariyya5 7.524236e+17 7/11/2016 8:45:39 AM
## 3      ã\201¡ããã\201 yuzuchaaan777 7.524236e+17 7/11/2016 8:45:38
AM
## 4               chutney  plainparatha 7.524236e+17 7/11/2016 8:45:38 AM
## 5 à¥\220 à¤à¤¾à¤°à¤¤ à¥\220     dharam_vj 7.524236e+17 7/11/2016 8:45:37
AM
## 6 Dipendra Dipzo Khati DipendraDipzo 7.524236e+17 7/11/2016 8:45:36 AM
##
tweets
## 1
ISIS influence on the decline as terrorists lose Twitter battles     - CNET
http://www.cnet.com/news/isis-influence-twitter-on-the-decline-us-state-
department/#ftag=CAD590a51e
## 2
RT @AyishaBaloch: #IndiaISISandBangladesh And world can ALSO not ignore the
truth revealing india 's role in providin explosive to ISIS httpâ\200¦
## 3
@Laika_isis @wink_BF
ã©ãªã¢ã\201£ã\201¦ã\201ªã\201ã\201ã\201ã\201
## 4
RT @KabirTaneja: Anti-ISIS volunteer fighting with the Kurds. things are
getting strange on planet Earth.  #PokemonGO https://t.co/ARdBQ4â\200¦
## 5 RT @MrsGandhi: It 's Urdu dailies not internet alone that 's turning
Muslims into terrorists #MustRead @tufailelif
http://www.dailyo.in/politics/muslims-radicalisation-isis-hyderabad-ramzan-
internet-war-of-badr-prophet-muhammad-orlando-shooting/story/1/11599.html
## 6 RT @MrsGandhi: It 's Urdu dailies not internet alone that 's turning
Muslims into terrorists #MustRead @tufailelif
http://www.dailyo.in/politics/muslims-radicalisation-isis-hyderabad-ramzan-
internet-war-of-badr-prophet-muhammad-orlando-shooting/story/1/11599.html
##          date
## 1 2016-11-07
## 2 2016-11-07
## 3 2016-11-07
## 4 2016-11-07
## 5 2016-11-07
## 6 2016-11-07
```

Tokenization

```
tweets$date<-as.Date(tweets$time, "%d/%m/%Y %H:%M:%S")
tidy_tweets<-tweets
%>%group_by(name,username,tweetid)%>%mutate(ln=row_number())%>%unnest_tokens(
word,tweets)%>%ungroup()
head(tidy_tweets,5)
```

```
## # A tibble: 5 x 7
##   name         username tweetid time                  date          ln word
##   <chr>        <chr>      <dbl> <chr>                 <date>     <int> <chr>
## 1 Sean Ferig~ ferigan  7.52e17 7/11/2016 8:45:3~ 2016-11-07     1 isis
## 2 Sean Ferig~ ferigan  7.52e17 7/11/2016 8:45:3~ 2016-11-07     1 influen~
## 3 Sean Ferig~ ferigan  7.52e17 7/11/2016 8:45:3~ 2016-11-07     1 on
## 4 Sean Ferig~ ferigan  7.52e17 7/11/2016 8:45:3~ 2016-11-07     1 the
## 5 Sean Ferig~ ferigan  7.52e17 7/11/2016 8:45:3~ 2016-11-07     1 decline
```

```
t_wrd<-tidy_tweets %>%count(word,sort=TRUE)
head(t_wrd,5)
```

```
## # A tibble: 5 x 2
##   word        n
##   <chr>  <int>
## 1 isis  113117
## 2 ã      93958
## 3 rt     86464
## 4 the    67765
## 5 à      58118
```

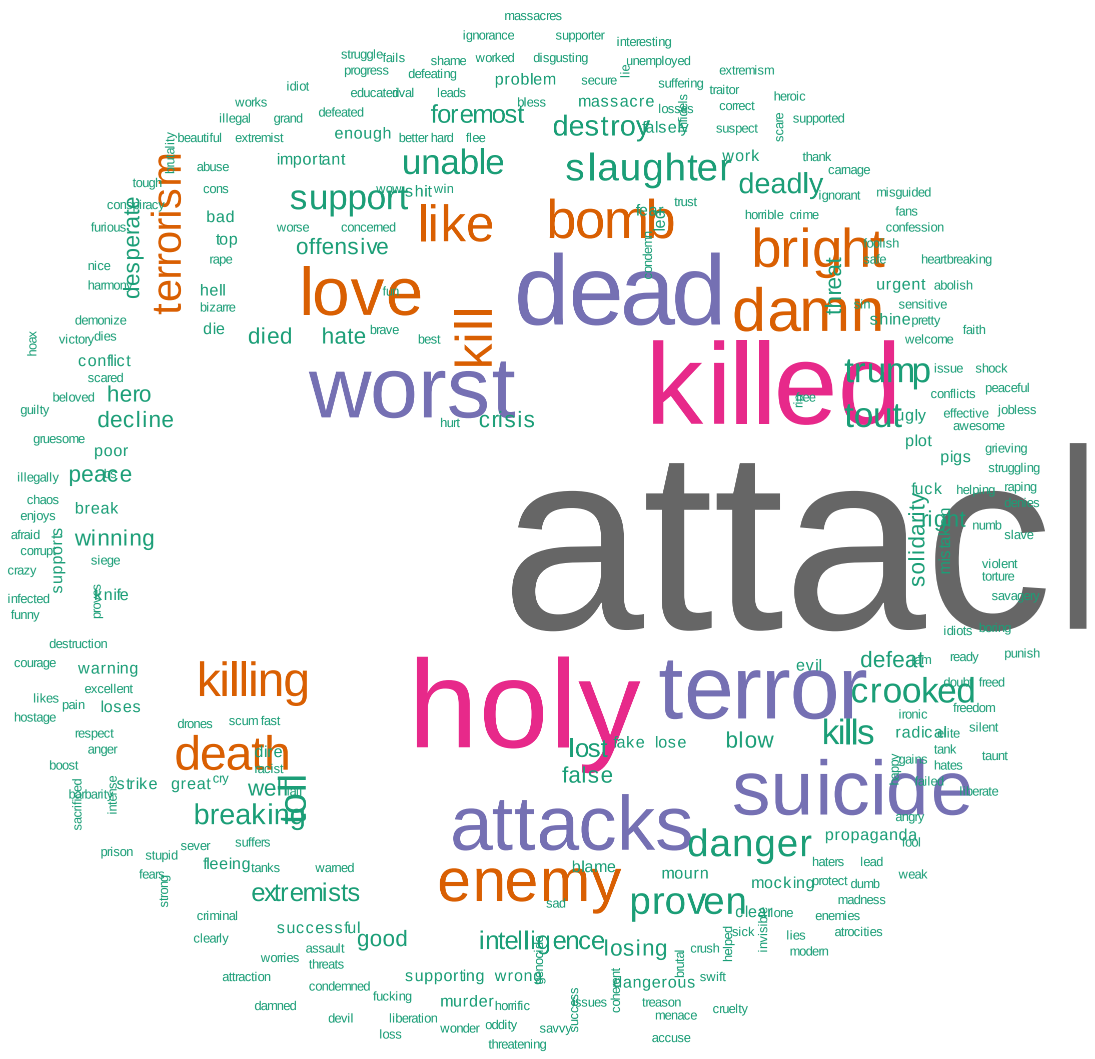
tidy_tweets contains all words like ‘the’,‘is’,‘are’ etc. So, we’ll take only the sentimental words by joining with lexicons.

Combining with Lexicons to get Sentiment

```
tweets_sentiment<-tidy_tweets%>% inner_join(get_sentiments("bing"))

## Joining, by = "word"
```

WordCloud of all words in ISIS tweets

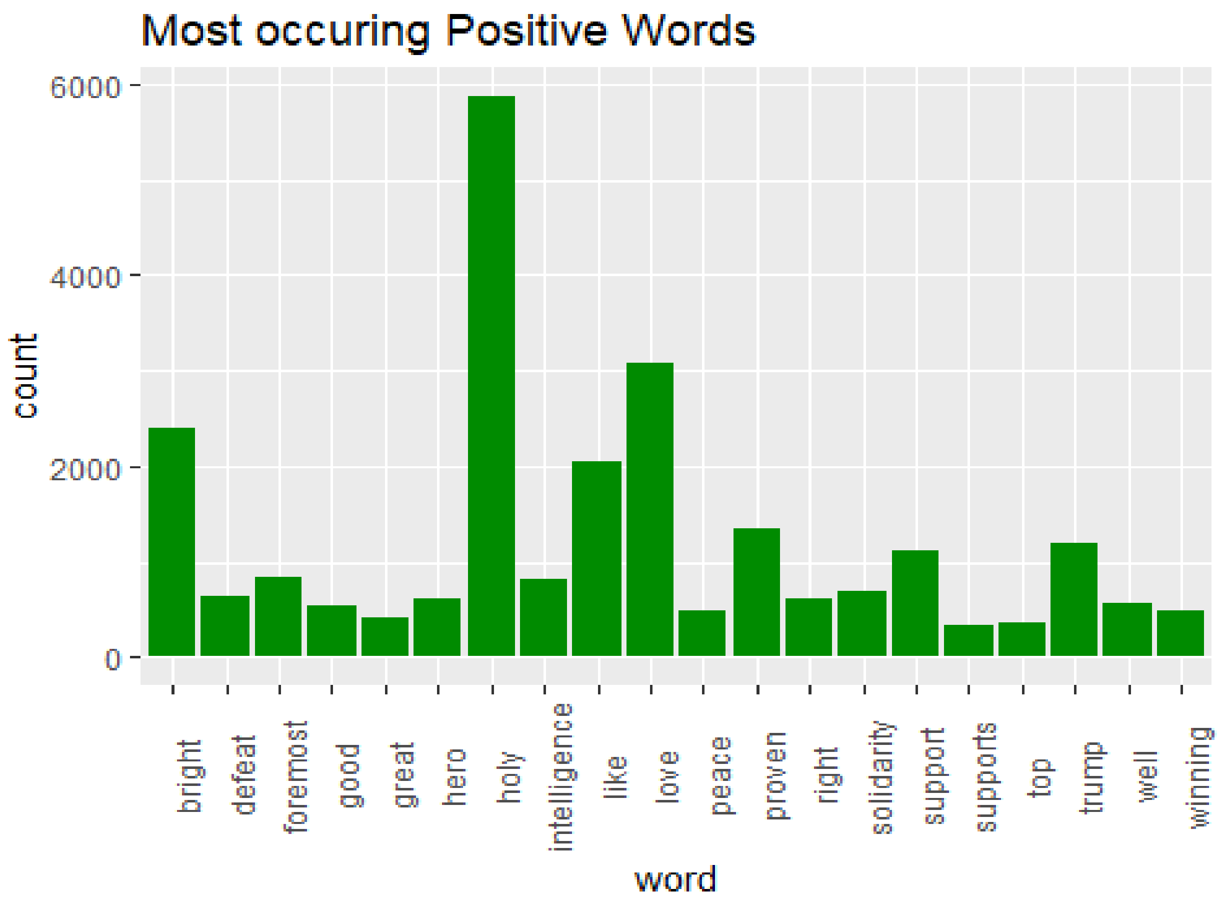
```
tot_wrd<-tweets_sentiment%>%count(word,sort=TRUE)
wordcloud(tot_wrd$word,tot_wrd$n, min.freq =5, scale=c(5, .2), random.order =
FALSE, random.color = FALSE,colors = brewer.pal(8, "Dark2"))
```



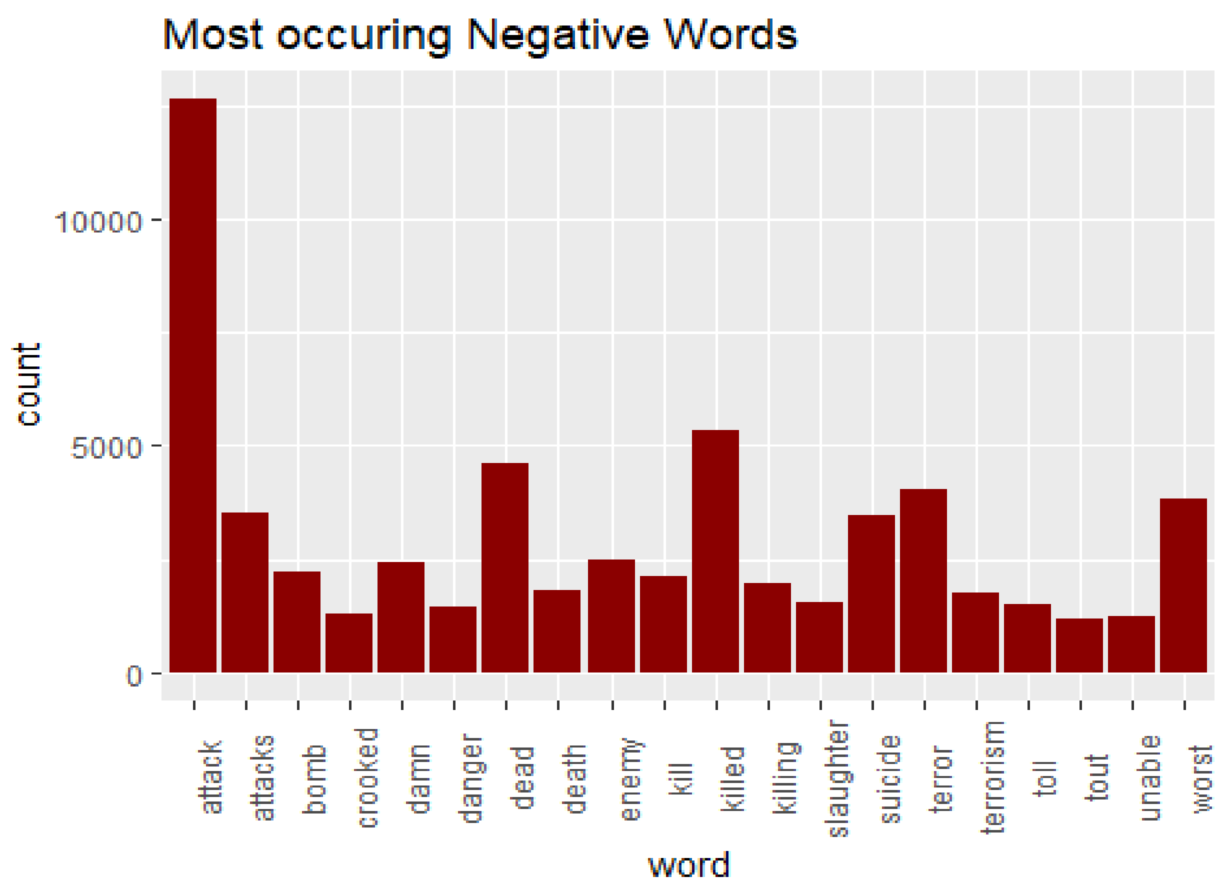
```
pos_neg<-tweets_sentiment %>%count(word,sentiment,sort=TRUE)
```

Plotting the most occuring positive and negative words.

```
pos_neg %>% filter(sentiment=='positive')%>%head(20)
%>%ggplot(aes(x=word,y=n))+geom_bar(stat="identity",fill="green4")+
  theme(axis.text.x=element_text(angle=90))+labs(title="Most occuring
Positive Words",y="count")
```

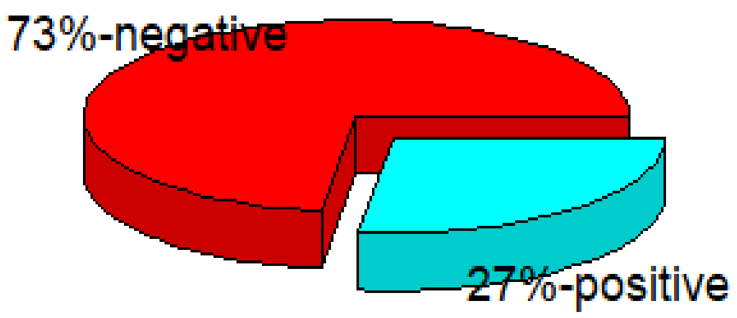


```
pos_neg %>% filter(sentiment=='negative')%>%head(20)
%>%ggplot(aes(x=word,y=n))+geom_bar(stat="identity",fill="red4")+
  theme(axis.text.x=element_text(angle=90))+labs(title="Most occuring
Negative Words",y="count")
```



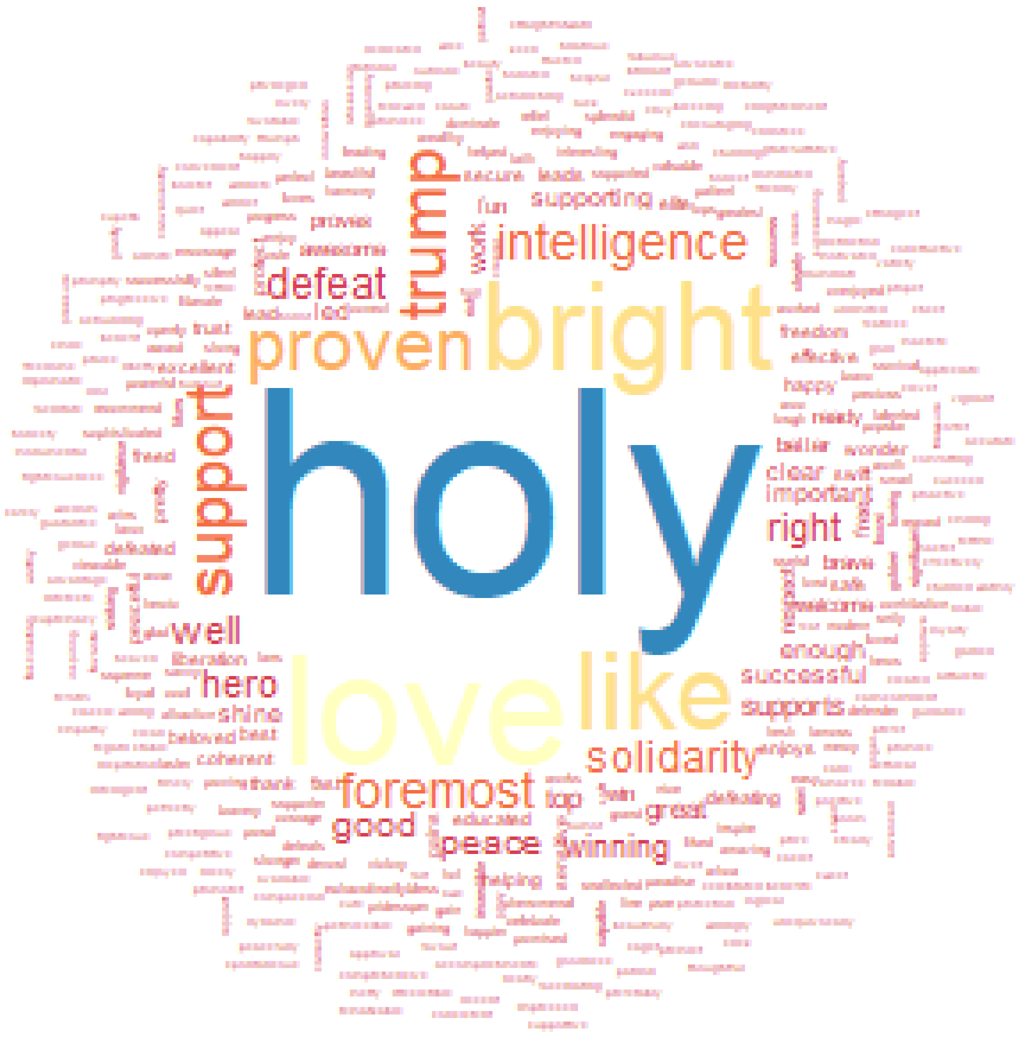
Percenatage of positive and negative words in the tweets.

```
perc<-
tweets_sentiment%>%count(sentiment)%>%mutate(total=sum(n))%>%group_by(sentime
nt)%>%mutate(percent=round(n/total,2)*100)%>%ungroup()
label <-
  c( paste(perc$percent[1],'%','-',perc$sentiment[1],sep=''),
     paste(perc$percent[2],'%','-',perc$sentiment[2],sep=''))

pie3D(perc$percent,labels=label,labelcex=1.1,explode=0.1     )
```

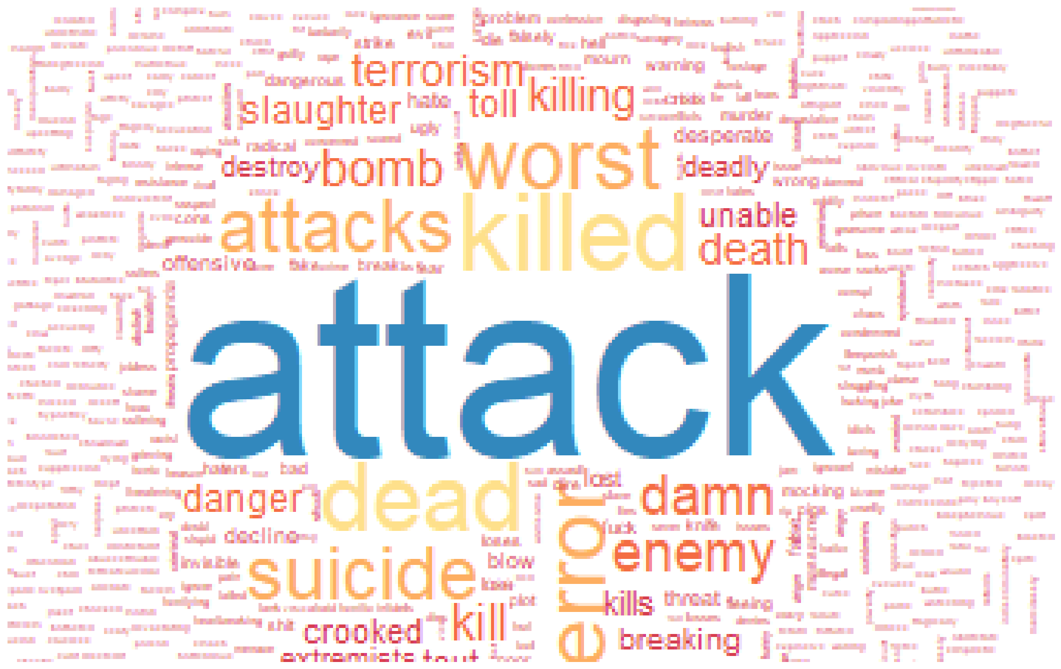


Wordcloud on Positive and Negative words

```
pos<-pos_neg %>% filter(sentiment=='positive')
neg<-pos_neg %>% filter(sentiment=='negative')
wordcloud(pos$word,pos$n, min.freq =5, scale=c(5, .2), random.order = FALSE,
random.color = FALSE,colors = brewer.pal(9,"Spectral"))
```

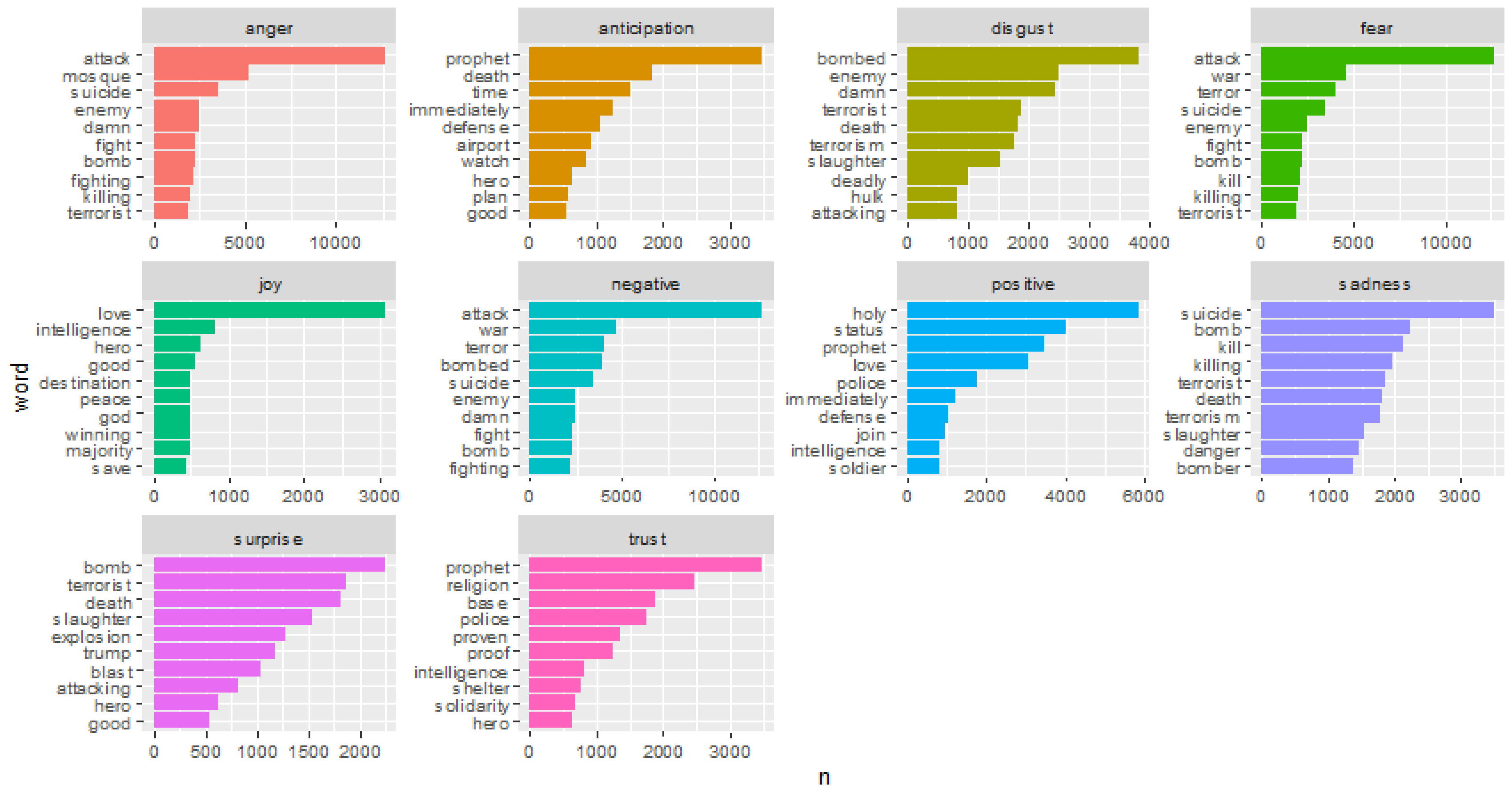


```
wordcloud(neg$word,neg$n, min.freq =5, scale=c(5, .2), random.order = FALSE,
random.color = FALSE,colors = brewer.pal(9, "Spectral"))
```



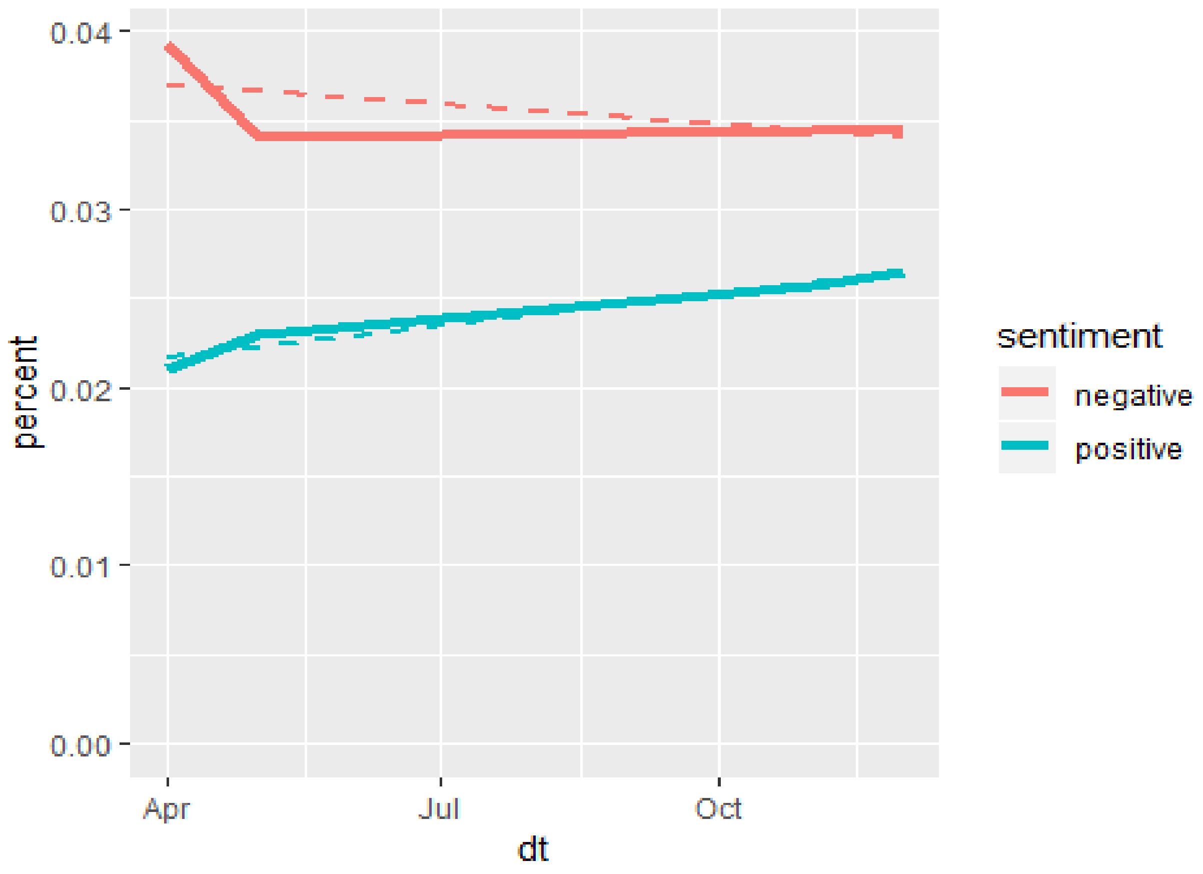
Top 10 words contributing to different sentiments

```
tidy_tweets%>%inner_join(get_sentiments("nrc")) %>%count(word,sentiment) %>%
group_by(sentiment)%>%top_n(10)%>%
  ungroup()
%>%mutate(word=reorder(word,n))%>%ggplot(aes(x=word,y=n,fill=sentiment)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ sentiment, scales = "free") +
  coord_flip()

## Joining, by = "word"

## Selecting by n
```


anger
attack
mosque
suicide
enemy
damn
fight
bomb
fighting
killing
terrorist
anticipation
prophet
death
time
immediately
defense
airport
watch
hero
plan
good
disgust
bombed
enemy
damn
terrorist
death
terrorism
slaughter
deadly
hulk
attacking
fear
attack
war
terror
suicide
enemy
fight
bomb
kill
killing
terrorist
joy
love
intelligence
hero
good
destination
peace
god
winning
majority
save
negative
attack
war
terror
bombed
suicide
enemy
damn
fight
bomb
fighting
positive
holy
status
prophet
love
police
immediately
defense
join
intelligence
soldier
sadness
suicide
bomb
kill
killing
terrorist
death
terrorism
slaughter
danger
bomber
surprise
bomb
terrorist
death
slaughter
explosion
trump
blast
attacking
hero
good
trust
prophet
religion
base
police
proven
proof
intelligence
shelter
solidarity
hero
word
n

Positive & Negative Words over Time

```
sentiment_by_time <- tidy_tweets %>%
  mutate(dt = floor_date(date, unit = "month")) %>%
  group_by(dt) %>%
  mutate(total_words = n()) %>%
  ungroup() %>%
  inner_join(get_sentiments("nrc"))

## Joining, by = "word"

sentiment_by_time %>%
  filter(sentiment %in% c('positive','negative')) %>%
  count(dt,sentiment,total_words) %>%
  ungroup() %>%
  mutate(percent = n / total_words) %>%
  ggplot(aes(x=dt,y=percent,col=sentiment,group=sentiment)) +
  geom_line(size = 1.5) +
  geom_smooth(method = "lm", se = FALSE, lty = 2) +
  expand_limits(y = 0)
```



N-grams

An n-gram is a sequence of n ``words'' taken, in order, from a body of text. If we n = 2, then we get a bigram.

```
demo_bigrams <- unnest_tokens(tweets, input = tweets, output = bigram, token
= "ngrams", n=2)
demo_bigrams %>%
  count(bigram, sort = TRUE)
```

```
## # A tibble: 336,681 x 2
##    bigram             n
##    <chr>          <int>
##  1 à à            49079
##  2 ã ã            43140
##  3 islamic state 10743
##  4 2016 07        10569
##  5 isis is         9489
##  6 ø ø             8012
##  7 ã ã            7980
##  8 ø ù             7240
##  9 https t.co      6973
## 10 ù ø             6658
## # ... with 336,671 more rows
```

```
bigrams_separated <- demo_bigrams %>%
  separate(bigram, c("word1", "word2"), sep = " ")
bigrams_filtered <- bigrams_separated %>%
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word)
```

New Bigram Counts

```
bigram_counts <- bigrams_filtered %>%
  count(word1, word2, sort = TRUE)
bigram_counts
```

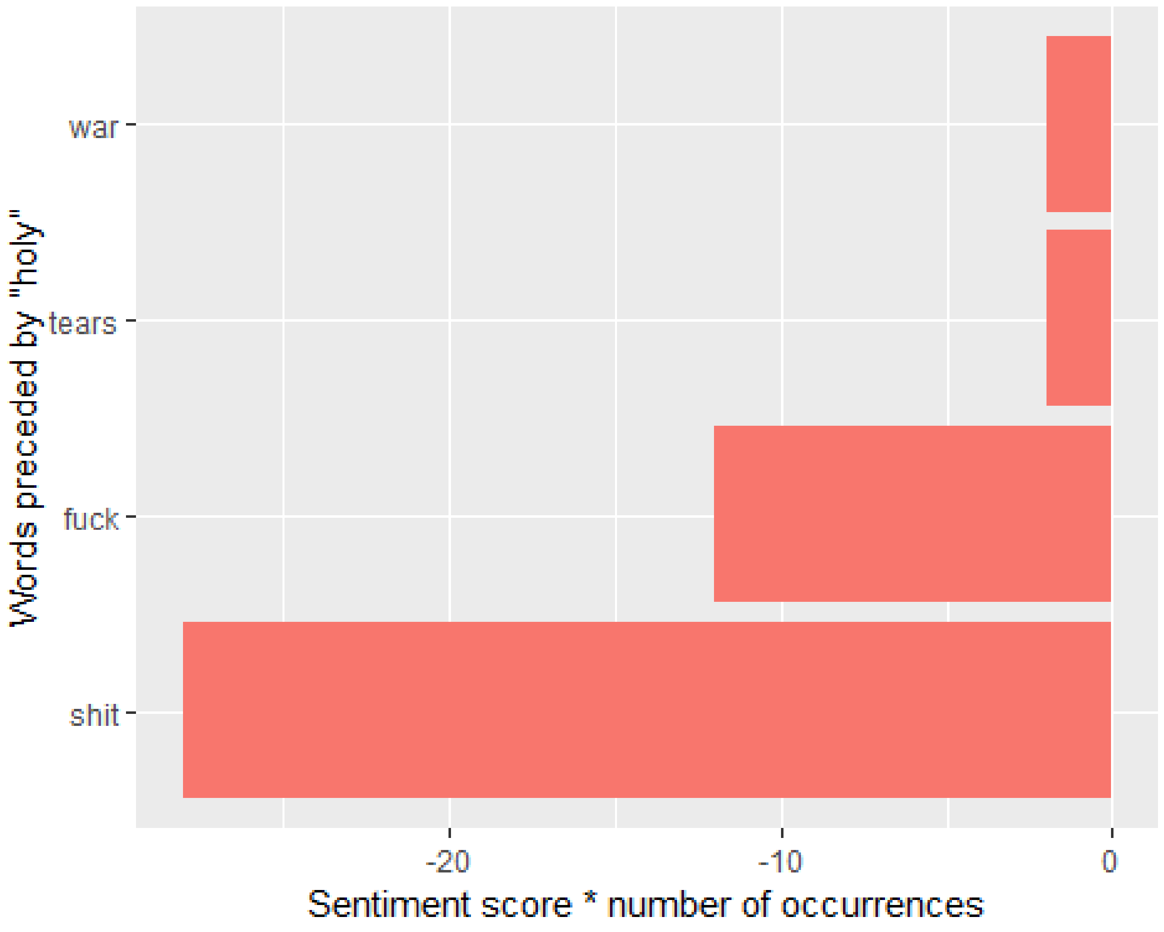
```
## # A tibble: 206,674 x 3
##    word1 word2     n
##    <chr> <chr> <int>
##  1 à     à     49079
##  2 ã     ã     43140
##  3 2016  07    10569
##  4 ø     ø      8012
##  5 ã     ã     7980
##  6 ø     ù      7240
##  7 https t.co   6973
##  8 ù     ø      6658
##  9 ã    ã      5950
## 10 å     ã      5617
## # ... with 206,664 more rows
```

Get AFINN Setiments

```
AFINN <- get_sentiments("afinn")
not_words <- bigrams_separated %>%
  filter(word1 == "holy") %>%
  inner_join(AFINN, by = c(word2 = "word")) %>%
  count(word2, score, sort = TRUE) %>%
  ungroup()
```

Get Sentiment Scores

```
not_words %>%
  mutate(contribution = n * score) %>%
  arrange(desc(abs(contribution))) %>%
  head(20) %>%
  mutate(word2 = reorder(word2, contribution)) %>%
  ggplot(aes(word2, n * score, fill = n * score > 0)) +
  geom_col(show.legend = FALSE) +
  xlab("Words preceded by \"holy\"") +
  ylab("Sentiment score * number of occurrences") +
  coord_flip()
```

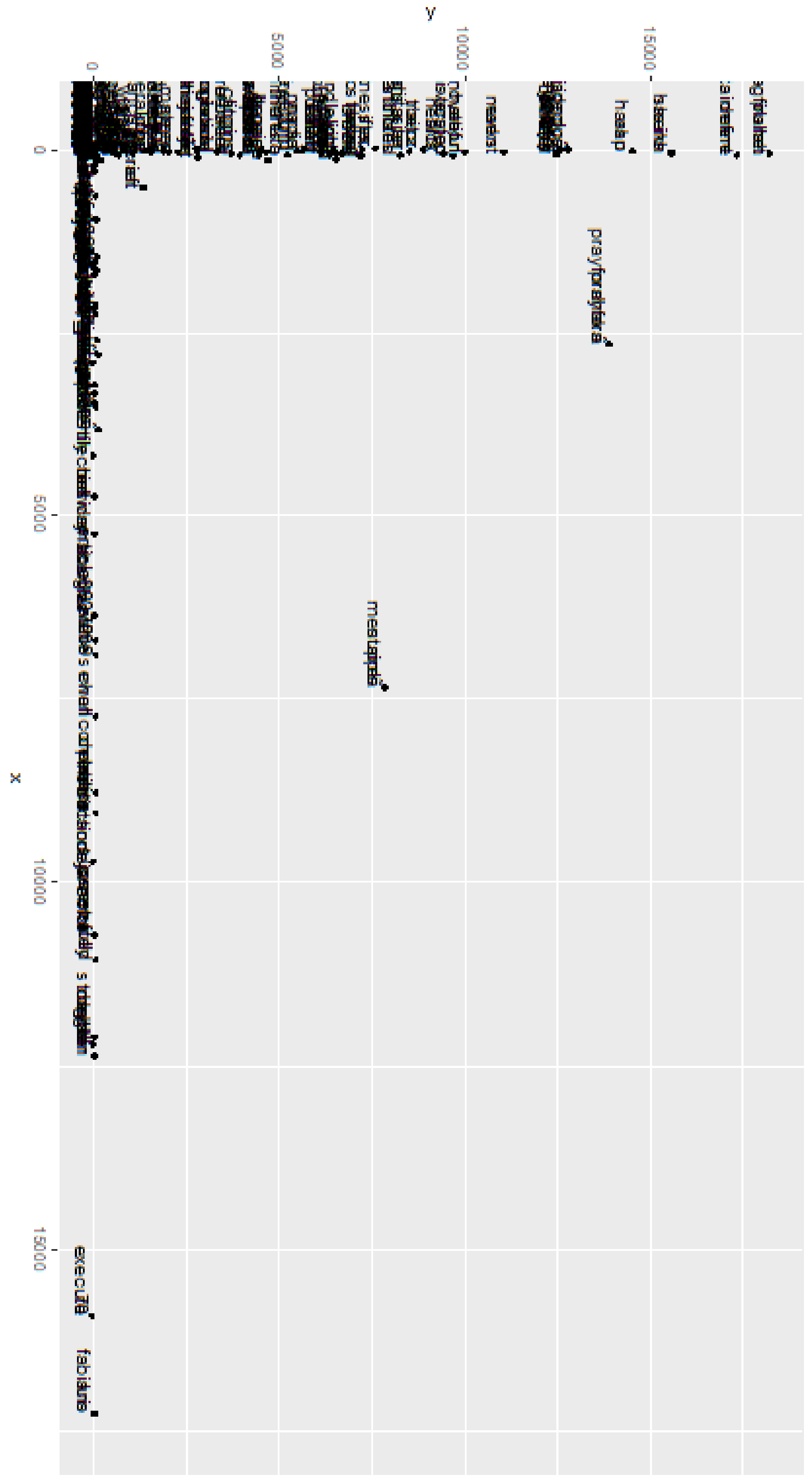


Plot Bigrams

```
bigram_graph <- bigram_counts %>%
  filter(n > 10) %>%
  graph_from_data_frame()

## Warning in graph_from_data_frame(.): In `d' `NA' elements were replaced
## with string "NA"

bigram_graph

## IGRAPH 147a33c DN-- 8600 17052 --
## + attr: name (v/c), n (e/n)
## + edges from 147a33c (vertex names):
##  [1] à      ->à                 ã      ->ã
##  [3] 2016   ->07                ø      ->ø
##  [5] ã      ->ã               ø      ->ù
##  [7] https  ->t.co              ù      ->ø
##  [9] ã     ->ã                 å      ->ã
## [11] https  ->twitter.com       ù      ->ù
## [13] holy   ->month             ã     ->ã
## [15] ã      ->å                 https  ->â
## + ... omitted several edges
```

Network of bigrams

```
set.seed(2017)
ggraph(bigram_graph, layout = "fr") +
  geom_edge_link() +
  geom_node_point() +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1)
```



y
0
5000
10000
15000
x
0
5000
10000
15000