Text Mining in R

Jarrod Griffin

1/14/2021

Importing Data

Tweets from the CCIDM Twitter page (https://twitter.com/CPP_CCIDM) were downloaded using the snscrape python package. Data was in the 'json' format, so we need to use the 'rjson' package to import it. All hashtags were removed manually. I will also generate IDs for each tweet.

```r
#install.packages("rjson")
#install.packages('tidyverse')
#install.packages('tidytext')
library('rjson')
library('tidyverse')
library('tidytext')

tweets <- fromJSON(file = 'CCIDM_tweets.json') %>%
  as_tibble() %>%           #tidytext package uses tibbles
  cbind(tweet_id = 31:1)    #generate IDs (tweets are in reverse chronological order)

head(tweets)
```

```
##
## 1                 We would like to thank CCIDM alumnus @WilliamAtienza2  for joining Center members I
## 2 We would like to thank everyone who participated in our workshops, attended our events, and helped
## 3
## 4                                                           Our full, six part Virtual R Workshop Ser
## 5                                                     Join the CCIDM, AMI, and Insights Association on Nov
## 6                                                                                Thank you to thos
##   tweet_id
## 1       31
## 2       30
## 3       29
## 4       28
## 5       27
## 6       26
```

Tidy Text Format and Tokenization

The tidy text format takes after Hadley Wickham's definition of tidy data, which is that:

- Each variable is a column
- Each observation is a row

- Each type of observational unit is a table

Tidy text is defined as a **table with one token per row**.

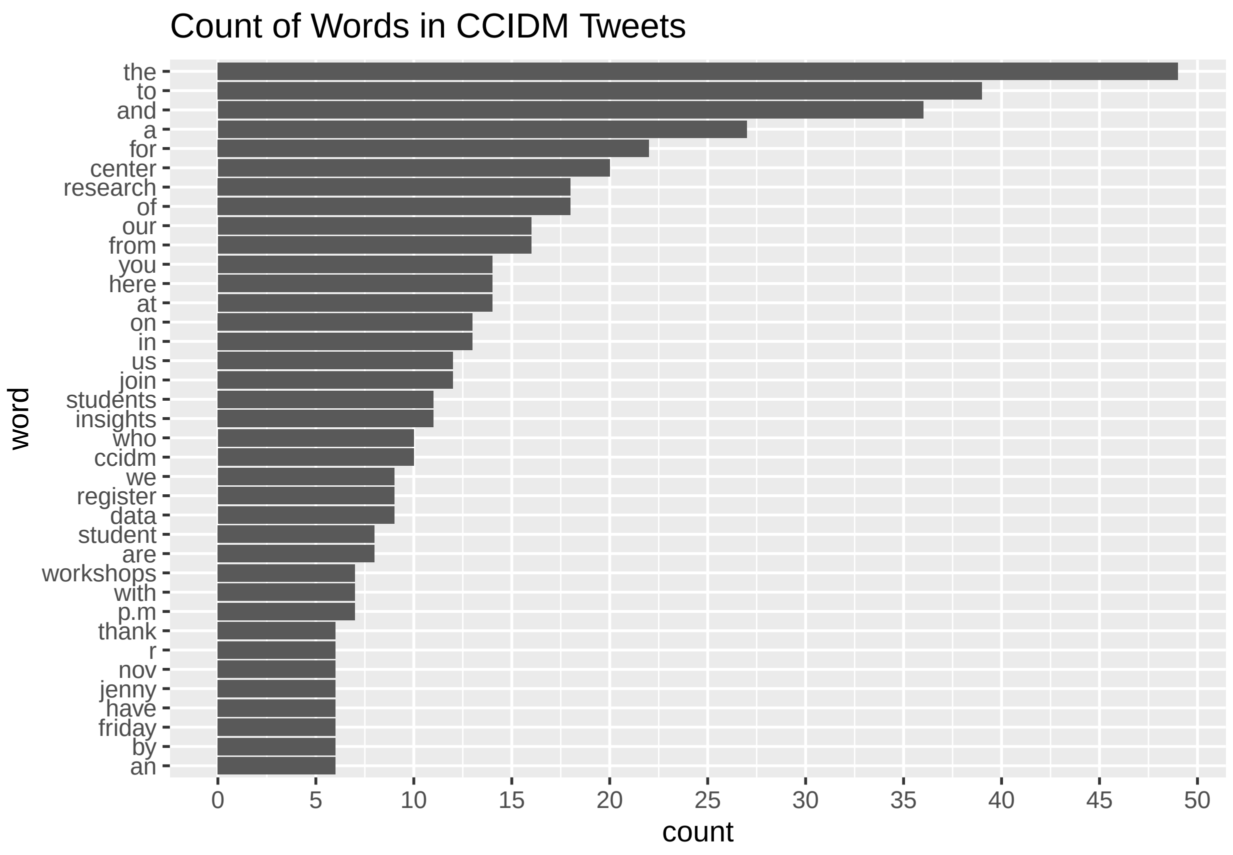
A token is defined as a **meaningful unit of text such as a word, sentence, paragraph or n-gram**.

The process of splitting the text up into tokens is called **tokenization**, and can be done by using the *unnest_tokens()* function.

```
tokenized_tweets <- unnest_tokens(tweets, input = 'tweet', output = 'word')
head(tokenized_tweets)
```

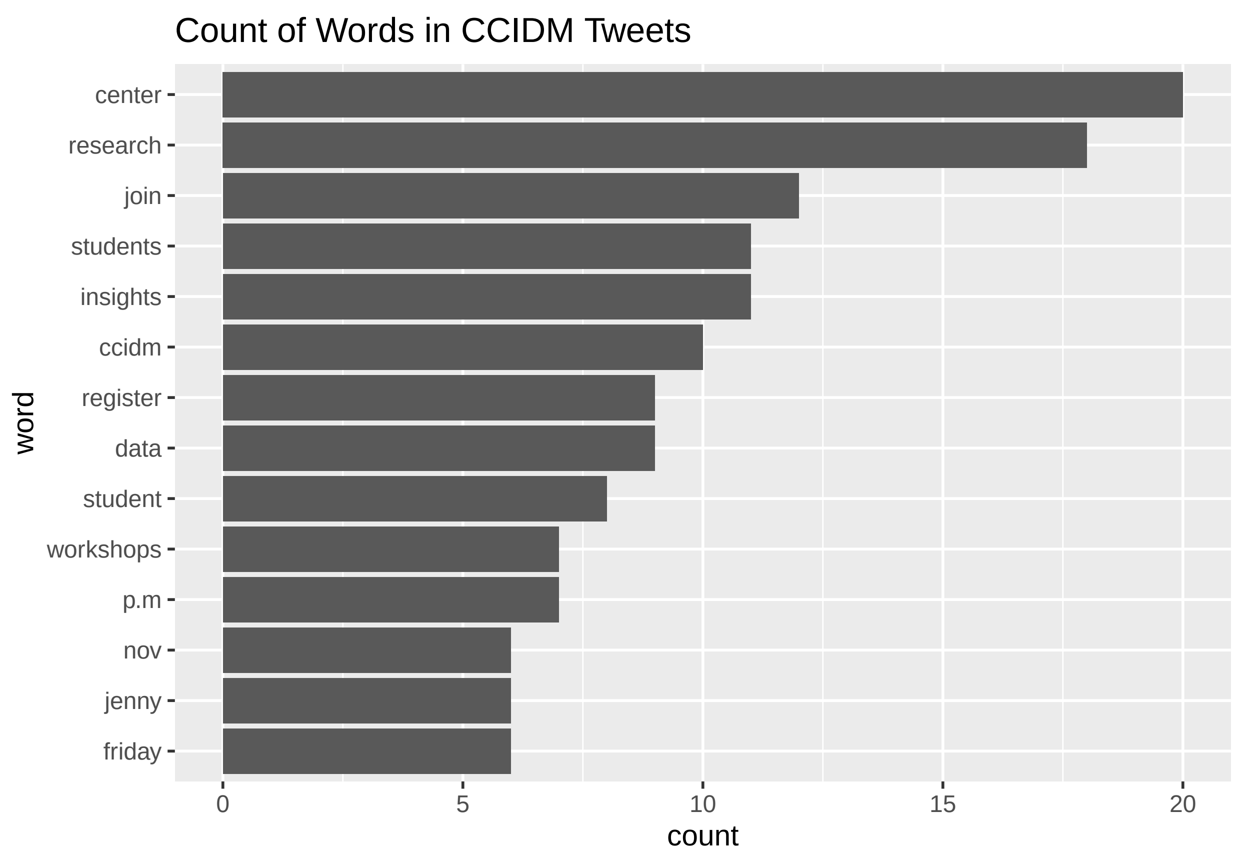
```
##     tweet_id  word
## 1         31    we
## 1.1       31 would
## 1.2       31  like
## 1.3       31    to
## 1.4       31 thank
## 1.5       31 ccidm
```

```
tokenized_tweets %>%
  count(word, sort = TRUE) %>%
  rename(count = n) %>%
  filter(count > 5) %>%
  mutate(word = reorder(word, count)) %>%
  ggplot(aes(x = count, y = word)) +
    geom_col()  +
    labs(title = "Count of Words in CCIDM Tweets") +
    scale_x_continuous(breaks = seq(0, 50, 5))
```



As you can see from the graph above, many of the words do not add value to our analysis. Words like "the", "and", or "to" are known as **stop words**. We will remove these stop words by calling the *anti_join(stop_words)* line of code. As you can see from the graph below, we have less words, but the words are much more interesting.

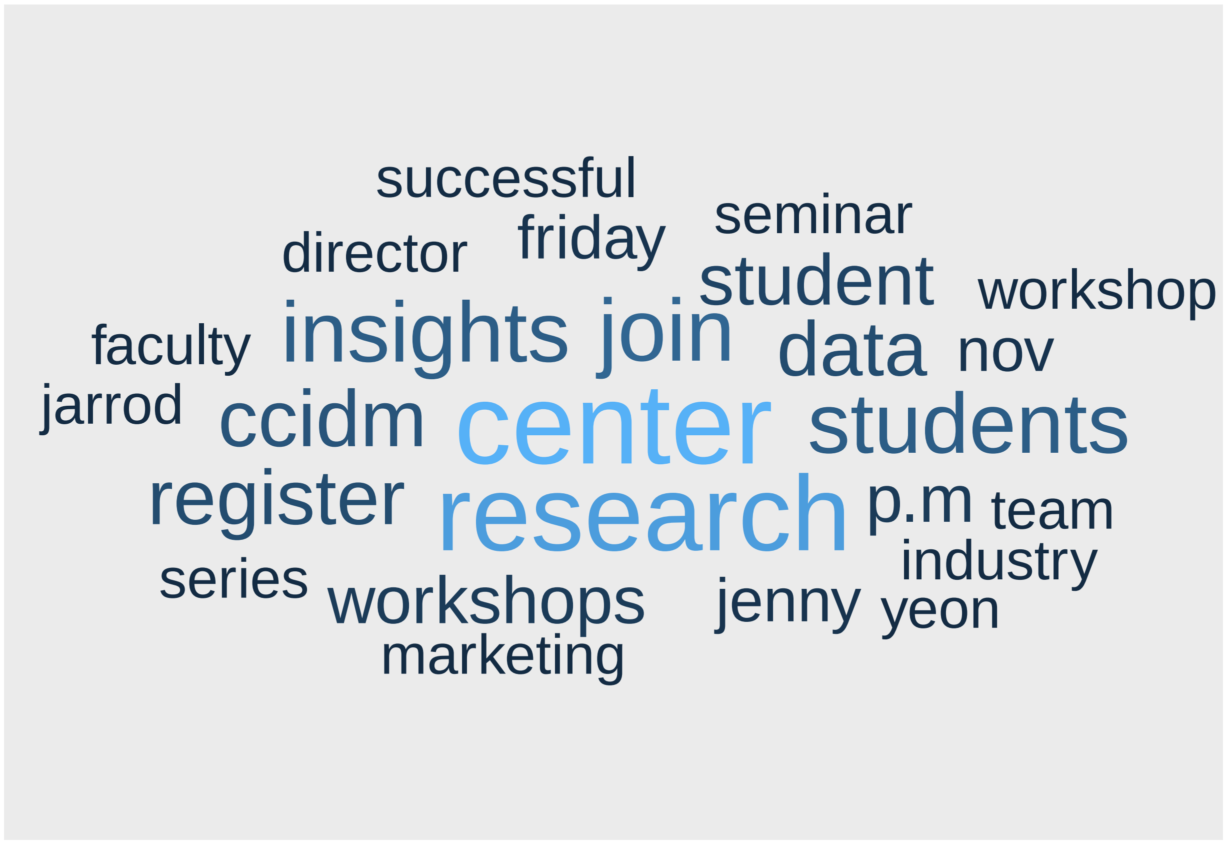
```
tokenized_tweets %>%
  anti_join(stop_words) %>% #finds where tweet words overlap with predefined stop words, and removes th
  count(word, sort = TRUE) %>%
  rename(count = n) %>%
  filter(count > 5) %>%
  mutate(word = reorder(word, count)) %>%
  ggplot(aes(x = count, y = word)) +
    geom_col() +
    labs(title = "Count of Words in CCIDM Tweets") +
    scale_x_continuous(breaks = seq(0, 50, 5))
```



There are many ways to visualize word counts, including word clouds as seen below.

```
#install.packages('ggwordcloud')
library('ggwordcloud')

tokenized_tweets %>%
  anti_join(stop_words) %>%
  count(word, sort = TRUE) %>%
  filter(n > 4) %>%
  ggplot(aes(label = word, size = n, color = n)) +
    geom_text_wordcloud() +
    scale_size_area(max_size = 15)
```



##Sentiment Analysis

When humans read text, we infer the emotional intent of the words. Sentiment analysis is the process of extracting these infered emotions from text. We can accomplish this by comparing the words in our text to words in many different sentiment lexicons. Lets take a look at some of these lexicons below. Some of these lexicons are subject to terms of use.

```
get_sentiments("afinn")    #integer value for positive/negative
```

```
## # A tibble: 2,477 x 2
##    word       value
##    <chr>      <dbl>
##  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 2,467 more rows
```

```
get_sentiments("bing")    #positive/negative
```

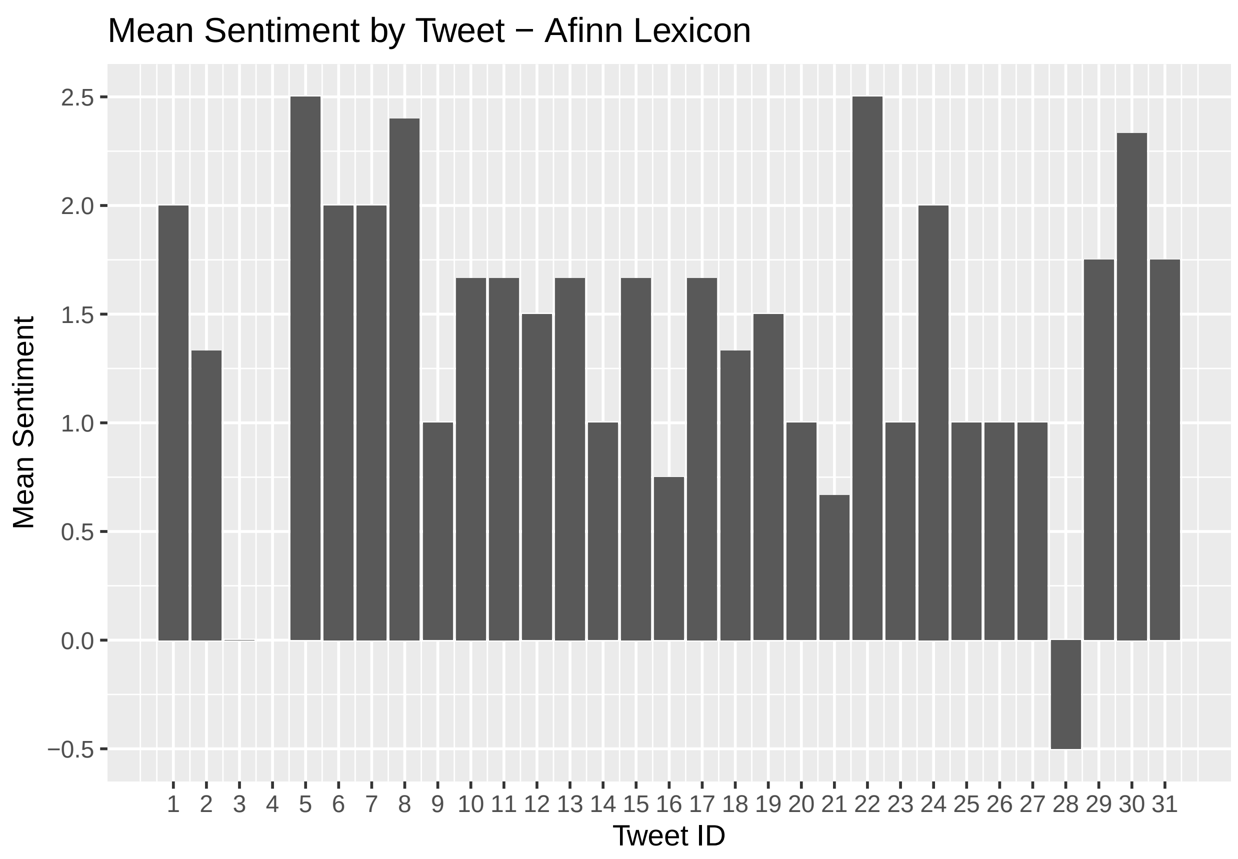
```
## # A tibble: 6,786 x 2
##    word        sentiment
##    <chr>       <chr>
##  1 2-faces     negative
##  2 abnormal    negative
##  3 abolish     negative
##  4 abominable  negative
##  5 abominably  negative
##  6 abominate   negative
##  7 abomination negative
##  8 abort       negative
##  9 aborted     negative
## 10 aborts      negative
## # ... with 6,776 more rows
```

```
get_sentiments("nrc")     #emotions
```

```
## # 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
```

There are thousands of words in each of the above lexicons. How do we see what words we have in our text overlap with what are in the lexicons? This can be accomplished by using the *inner_join()* function. Let's explore the three packages with some visualizations below.

```
tokenized_tweets %>%
  group_by(tweet_id) %>%
  inner_join(get_sentiments("afinn")) %>%
  summarise(mean_sentiment = mean(value)) %>%
  ggplot(aes(x = tweet_id, y = mean_sentiment)) +
    geom_col() +
    labs(title = 'Mean Sentiment by Tweet - Afinn Lexicon', x = "Tweet ID", y = 'Mean Sentiment') +
    scale_x_continuous(breaks = seq(1, 31)) +
    scale_y_continuous(breaks = seq(-1, 3, 0.5))
```



Looking at the chart above, we notice that it appears two tweets have a mean sentiment of 0. This is actually incorrect. Only the third tweet has a mean sentiment of 0, tweet 4 actually should be reported as an NA value. This is because there was no overlap between tweet 4 and the lexicon we used, meaning that no words were found to have any sentiment according to the Afinn lexicon. Let's confirm this below.

```
print("Tweet 4 words found in the Afinn lexicon should appear below: ")
```

```
## [1] "Tweet 4 words found in the Afinn lexicon should appear below: "
```

```
tokenized_tweets %>%
  filter(tweet_id==4)%>%
  inner_join(get_sentiments("afinn"))
```

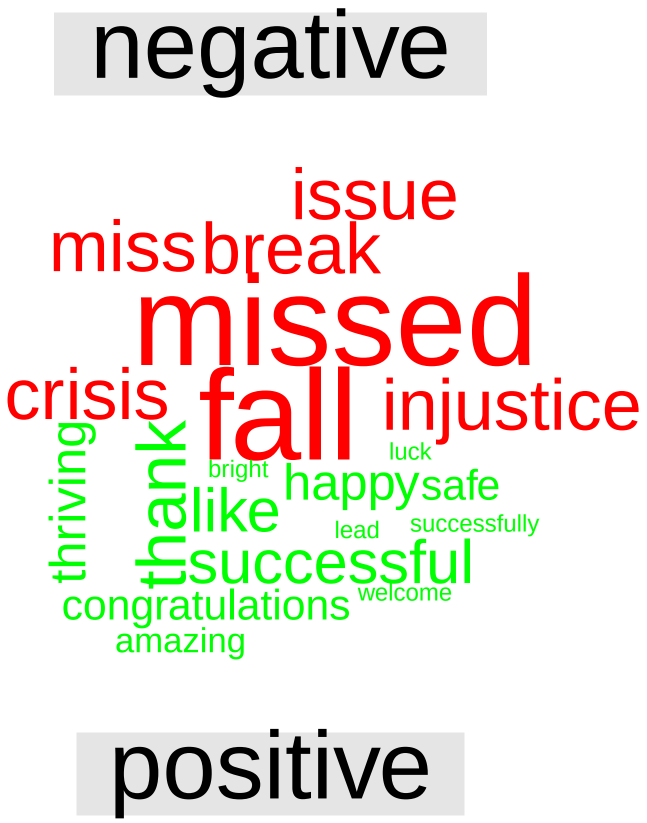
```
## [1] tweet_id word     value
## <0 rows> (or 0-length row.names)
```

No words were found in the 4th tweet AND the Afinn lexicon.

Lets take a look at the bing lexicon. The bing lexicon groups words into two sentiment categories, positive and negative. Lets plot our tweets into a word cloud to get a nice visual of our data.

```
#install.packages('reshape2')
#install.packages('wordcloud')
library('reshape2')
library('wordcloud')
```

```
tokenized_tweets %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>% #cast into matrix, grouped by neg and pos
  comparison.cloud(colors = c("red", "green"),
                   max.words = 20)
```



Term Frequency(tf) and Inverse Document Frequency (idf)

A common question in text mining is: What is this text about? There are a few ways to determine this, two of which are Term Frequency and Inverse Document Frequency.

- Term Frequency was discussed previously, and is a more simple way to determine what a text is about.
- Inverse Document Frequency is the implementation for Zipf's law stating that the frequency that a word appears is inversely pororptional to its rank/importance. That is, the less a word shows up in a text, higher its importance rank.

Below we see the standard tf (Term Frequency) for all of the CCIDM tweets.

```
tokenized_tweets %>%
  count(word, sort = TRUE) %>%
  rename(count = n) %>%
  head()
```

```
##     word count
## 1    the    49
## 2     to    39
## 3    and    36
## 4      a    27
## 5    for    22
## 6 center    20
```

Below we see the entire TF-IDF dataframe. We are most interested in the *tf_idf* column, as that will provide us the weighted rank/importance for our text.

```
tweet_tf_idf <- tokenized_tweets %>%
  count(word, tweet_id, sort = TRUE) %>%
  rename(count = n) %>%
  bind_tf_idf(word, tweet_id, count)

head(tweet_tf_idf)
```

```
##   word tweet_id count         tf       idf     tf_idf
## 1  and       16     4 0.10256410 0.3429448 0.03517382
## 2 team        2     4 0.09523810 2.7408400 0.26103238
## 3  the        6     4 0.10256410 0.1017827 0.01043925
## 4  the        7     4 0.10000000 0.1017827 0.01017827
## 5    a       17     3 0.09090909 0.6007739 0.05461581
## 6  and       27     3 0.08333333 0.3429448 0.02857873
```

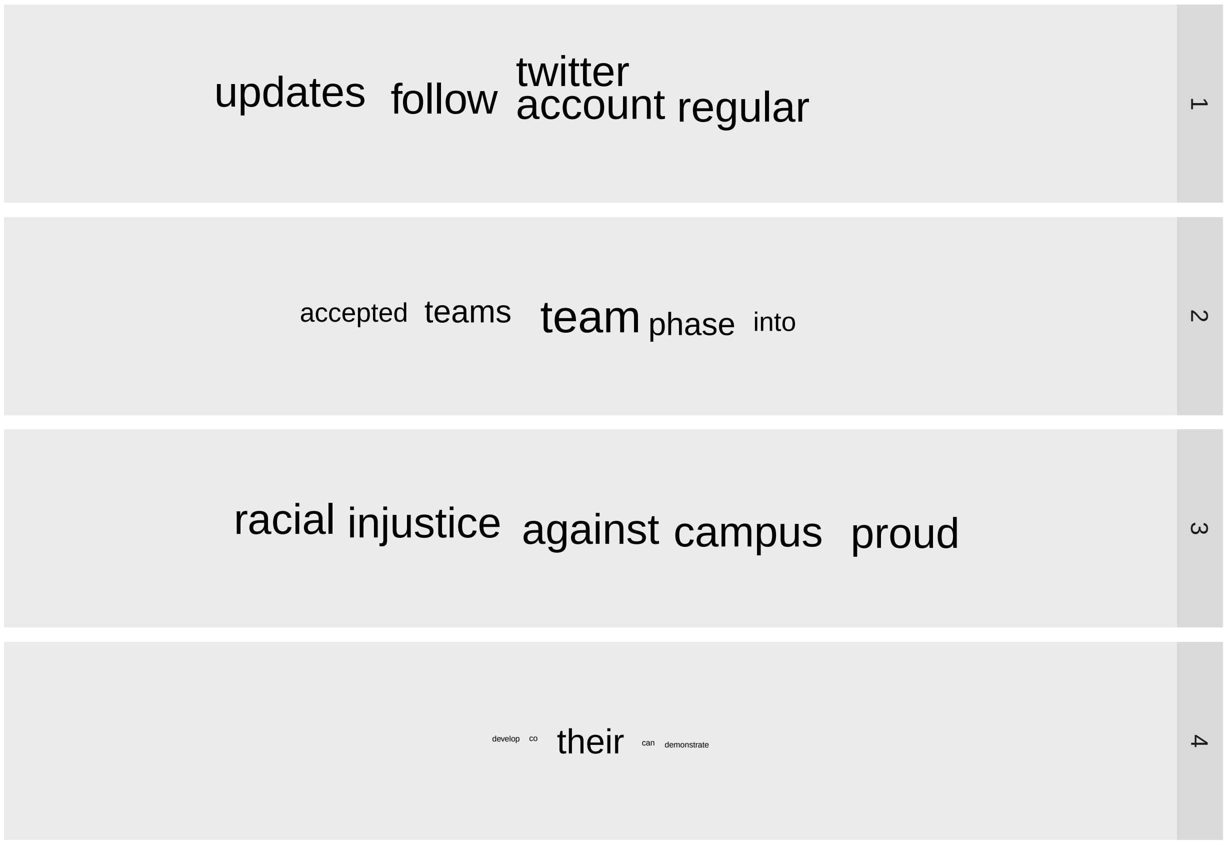
The above data frame does not really show us the most important words. Lets group the words from each tweet and order them to find the most important words.

```
tweet_tf_idf %>%
  select(word, tweet_id, tf_idf) %>%
  group_by(tweet_id) %>%
  slice_max(order_by = tf_idf, n = 5, with_ties=FALSE) %>% #takes top 5 words from each tweet
  filter(tweet_id < 5) %>% #just look at 4 tweets
  ggplot(aes(label = word, size=tf_idf)) +
    geom_text_wordcloud() +
    facet_grid(rows = vars(tweet_id))
```

```
## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=8, height=8' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=4, height=8' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=8' are unlikely values in pixels
```


updates
follow
twitter
account
regular
1
accepted
teams
team
phase
into
2
racial
injustice
against
campus
proud
3
develop
co
their
can
demonstrate
4