Text Mining in R

Jarrod Griffin

1/14/2021

Before we begin, please run the code below to install all packages we will be using today.

```
packages = c(
  'rtweet',
  'httpuv',
  'tidyverse',
  'rtweet',
  'tidytext',
  'ggwordcloud',
  'reshape2',
  'wordcloud',
  'igraph',
  'ggraph',
  'topicmodels',
  'tm'
)

package.check <- lapply( #by vikram
  packages,
  FUN = function(x) {
    if (!require(x, character.only = TRUE)) {
      install.packages(x, dependencies = TRUE)
    }
  }
)
```

If you have access to the Twitter API, replace the following token and key strings with the ones generated from your Twitter developer account.

```
library('rtweet')
library('httpuv')

api_key <- "8GwQcGEWxLAbcHvJBbdlqu7Xf"
api_secret_key <- "tzsZNg9rvDNqrCA3Btc4Gka9iv6W9NkSxo2WfSwsGZIHm8Rnk3"
app_name <- "CenterScrape"
access_token <- "635181580-sD7F6vWdH7gt8kPHh4i90HolDX4aWZNhFZ8Y9DV7"
access_token_secret<-"ZmtuRto1MYnSuUwkPeANh8MVkFcjCtA4YMv0MNooZUhvE"

token <- create_token(
  app = app_name,
  consumer_key = api_key,
```

```
  consumer_secret = api_secret_key,
  access_token = access_token,
  access_secret = access_token_secret
)

#get_token()
```

Importing Data

Tweets from the CCIDM Twitter page (https://twitter.com/CPP_CCIDM) were downloaded using the Twitter API. If you do not have access to the Twitter API, see the *Twitter API Access* instructions.

Let's use the *get_timelines()* function from the rtweet package to get all tweets on the CCIDM Twitter timeline. The function will return a lot of information, so lets just select some relevant columns. Our goal is to end up with just the text tweet data from the CCIDM twitter account, meaning we want to exclude retweets. We will also strip links from each tweet. We can also generate a .csv of the tweets.

```
library('tidyverse')
library('rtweet')
timelineDF <- get_timelines('CPP_CCIDM')

removeURL <- function(x) gsub("http[[:alnum:][:punct:]]*", "", x)

numTweets <- timelineDF %>%
  filter(is_retweet == FALSE) %>%
  nrow()

tweets <- timelineDF %>%
  filter(is_retweet == FALSE) %>%
  select(text) %>%
  cbind(tweet_id = numTweets:1) %>%
  rename(tweet = text) %>%
  mutate(tweet = removeURL(tweet))

write_csv(tweets, "tweets.csv")

head(tweets)
```

```
##
## 1                                              Join us on Friday, March 5th from 1 - 2pm for the next
## 2     Thank you Erantzeri Corona for his informational talk about the importance of developing data s
## 3                                         Join us TOMORROW to hear VP of Digital Marketing and eComme
## 4                                          Come hear the VP of Digital Marketing and eCommerce at Vik
## 5                               Come hear the VP of Digital Marketing and eCommerce at Viking Cruises, E
## 6 We would like to thank CCIDM alumnus @WilliamAtienza2  for joining Center members last Friday to di
##   tweet_id
## 1       36
## 2       35
## 3       34
## 4       33
## 5       32
## 6       31
```

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**.

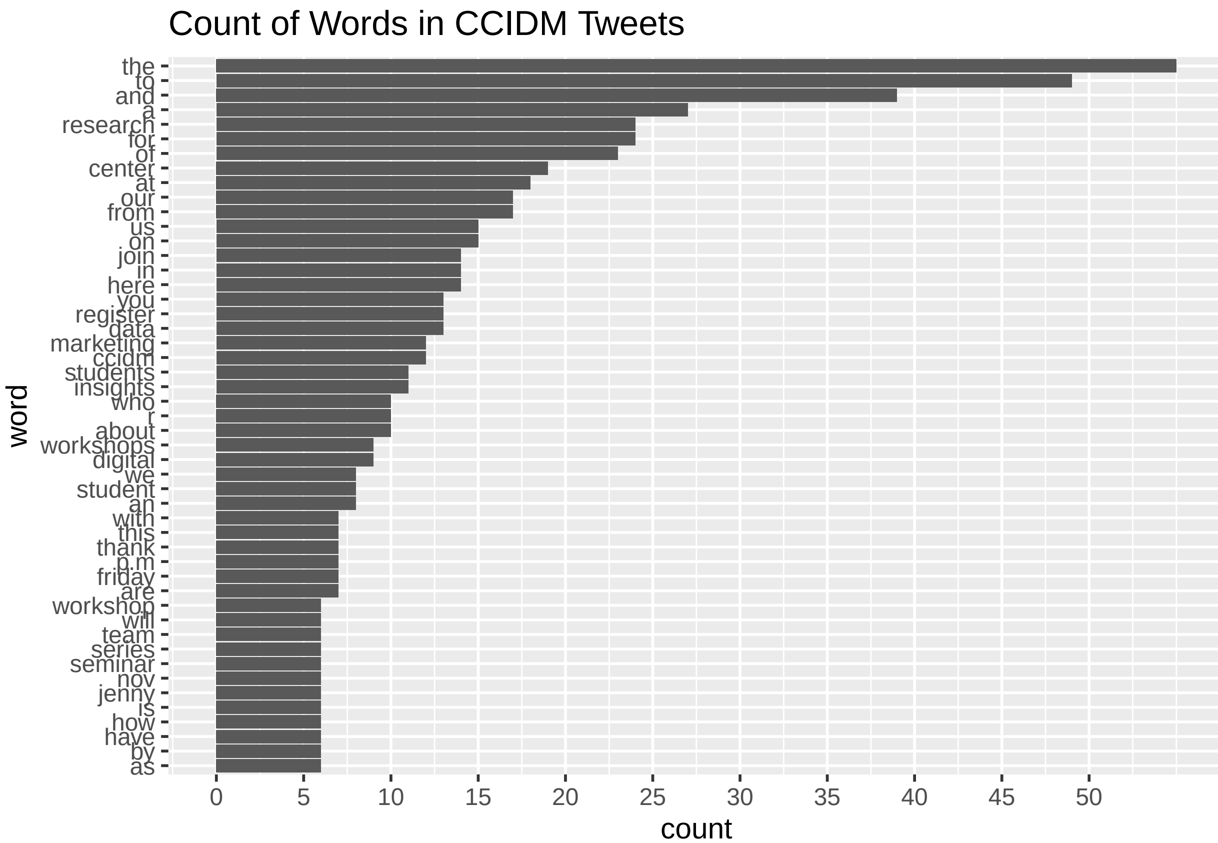
A token is defined as a **meaningful unit of text such as a word, sentence, paragraph or n-gram**.

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

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

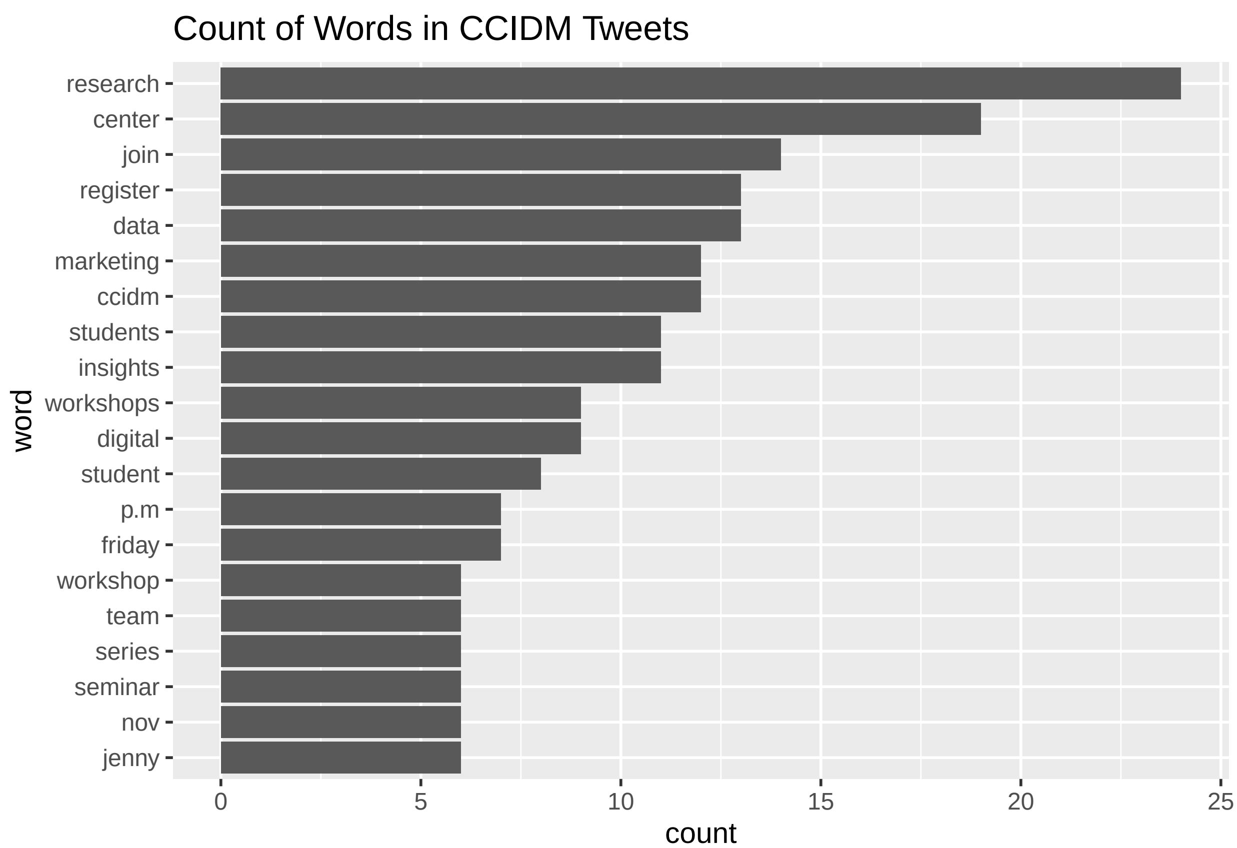
```
##   tweet_id   word
## 1       36   join
## 2       36     us
## 3       36     on
## 4       36 friday
## 5       36  march
## 6       36    5th
```

```
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.

```
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 inferred 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
```

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

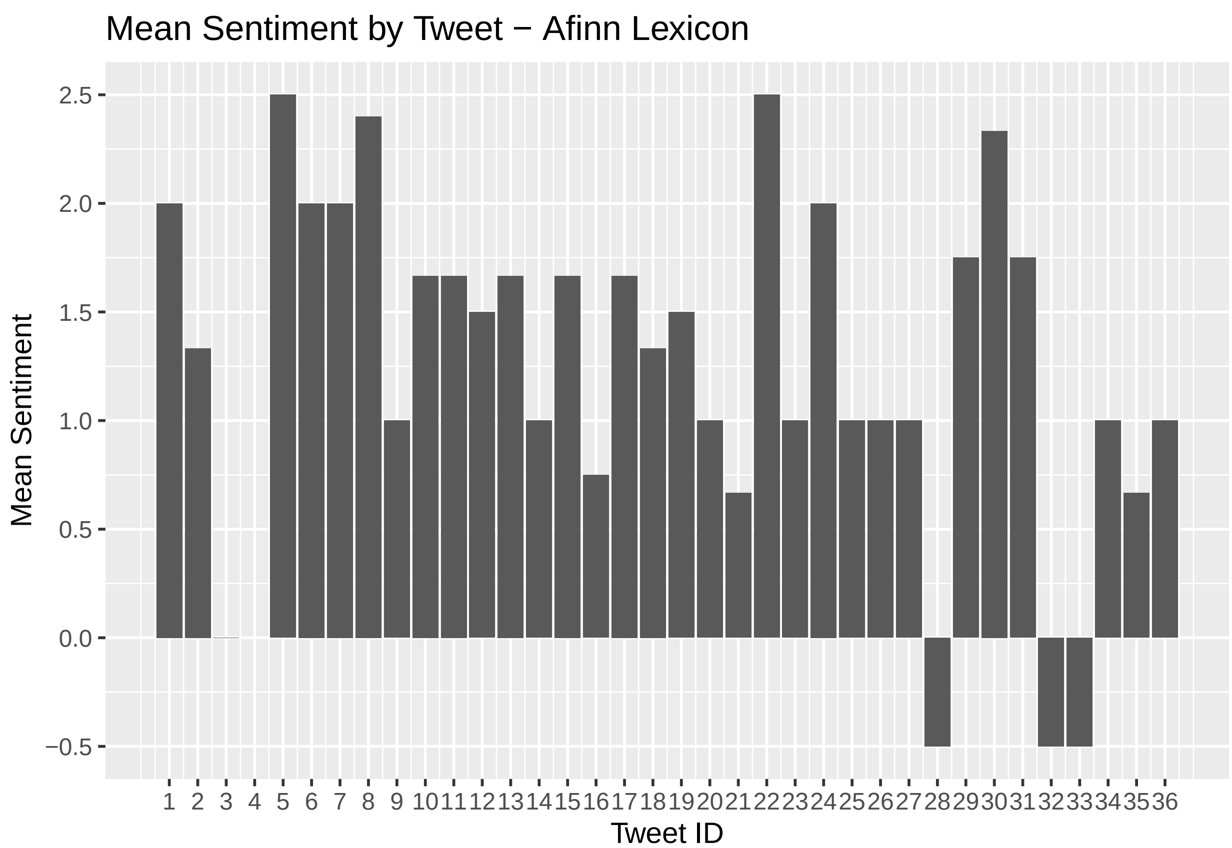
```
## # A tibble: 6,786 x 2
##    word        sentiment
##    <chr>       <chr>
##  1 2-faces     negative
##  2 abnormal    negative
##  3 abolish     negative
##  4 abominable  negative
##  5 abominably  negative
##  6 abominate   negative
##  7 abomination negative
##  8 abort       negative
## 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

```r
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.

```r
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, numTweets)) +
    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 also take a look at tweet_id number 28 as it seems to have some negative sentiment, according to the Afinn lexicon.

```
tweets[28, 1]
```

```
## [1] "Remember to join us tomorrow at noon for the first of our R workshops. If you attend every works
```

As you can see from above, the tweet isn't really negative. The above highlights some possible errors when using lexicons.

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.

```
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 is the count of a token divided by the total number of tokens.
- 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) %>%
  mutate(total=sum(count))%>%
  mutate(tf=count/total) %>%
  head()
```

```
##       word count total         tf
## 1      the    55  1340 0.04104478
## 2       to    49  1340 0.03656716
## 3      and    39  1340 0.02910448
## 4        a    27  1340 0.02014925
## 5      for    24  1340 0.01791045
## 6 research    24  1340 0.01791045
```

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.

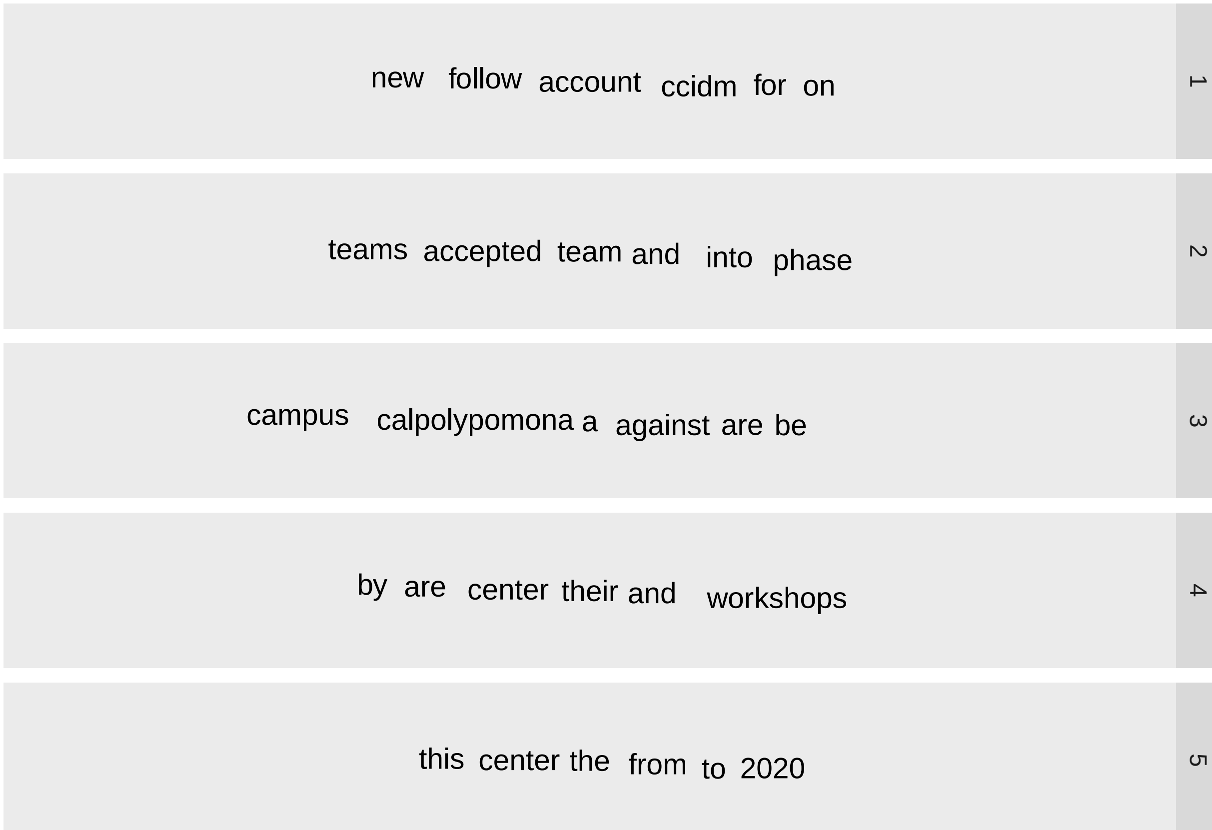
```
tweet_tf_idf <- tokenized_tweets %>%
  count(word, tweet_id, sort = TRUE) %>%
  rename(count = n) %>%
  bind_tf_idf(word, tweet_id, count)

head(tweet_tf_idf)
```

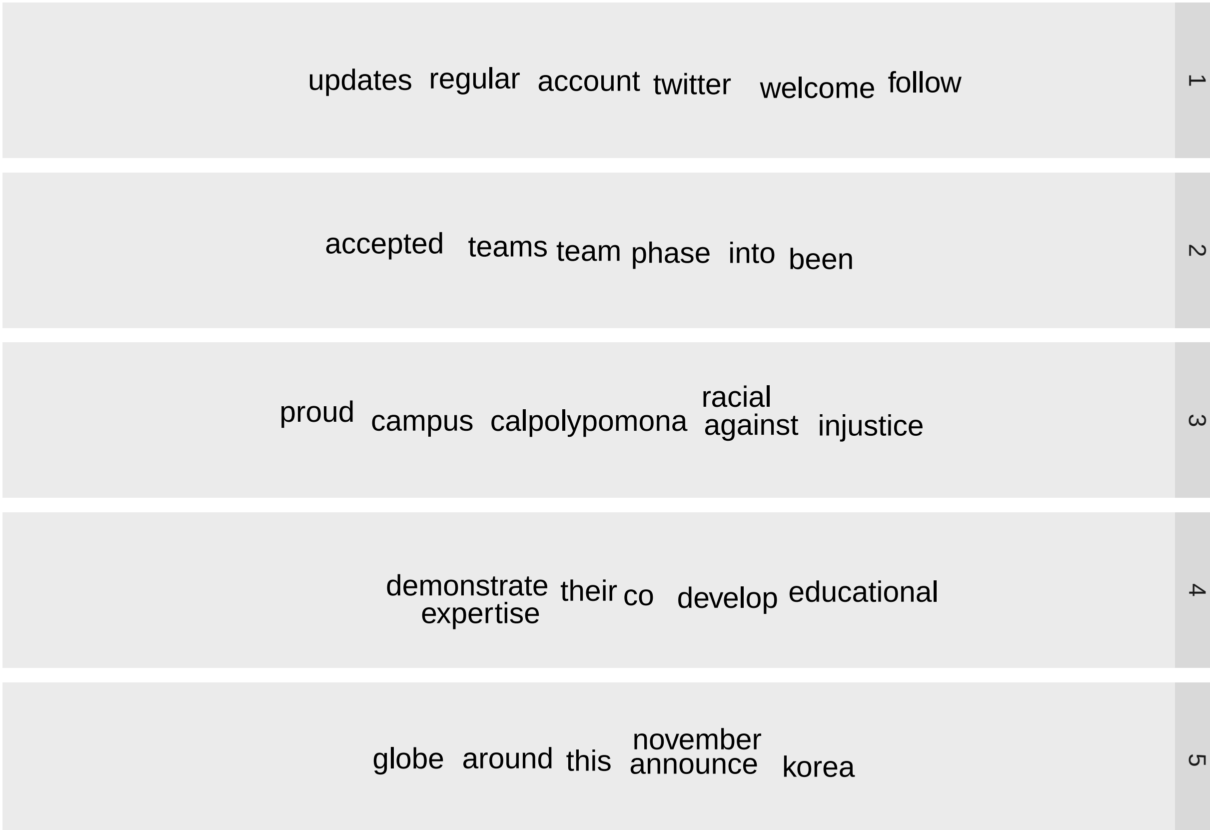
```
##   word tweet_id count         tf       idf     tf_idf
## 1  and       16     4 0.10000000 0.3646431 0.03646431
## 2 team        2     4 0.09523810 2.8903718 0.27527350
## 3  the        6     4 0.10526316 0.1177830 0.01239821
## 4  the        7     4 0.10000000 0.1177830 0.01177830
## 5    a       17     3 0.08823529 0.7503056 0.06620343
## 6  and       27     3 0.07894737 0.3646431 0.02878761
```

Simple counts of word frequencies can be misleading and not helpful in getting a good idea of your data. Lets demonstrate that below.

```
tweet_tf_idf %>%
  select(word, tweet_id, tf_idf, count) %>%
  group_by(tweet_id) %>%
  slice_max(order_by = count, n = 6, with_ties=FALSE) %>% #takes top 5 words from each tweet
  filter(tweet_id < 6) %>% #just look at 5 tweets
  ggplot(aes(label = word)) +
    geom_text_wordcloud() +
    facet_grid(rows = vars(tweet_id))
```




```
tweet_tf_idf %>%
  select(word, tweet_id, tf_idf) %>%
  group_by(tweet_id) %>%
  slice_max(order_by = tf_idf,n = 6, with_ties=FALSE) %>% #takes top 5 words from each tweet
  filter(tweet_id < 6) %>% #just look at 5 tweets
  ggplot(aes(label = word)) +
    geom_text_wordcloud() +
    facet_grid(rows = vars(tweet_id))
```



As you can see from above, the second set of word clouds provide us with much more interesting and relevant words. The second set of word clouds more accurately displays the important words in the tweet.

Relationship Between Words

So far we have only looked at words individually, and how those words relate to sentiment or frequency in document. But what if we want to know about how words relate to each other in a text? This can be accomplished though n-grams, where n is a number.

Previously we had tokenized by single words, but we can also tokenize by n number of words. Lets create bigrams from all of the tweets, then count and sort them.

```
tweets_bigram <- tweets %>%
  unnest_tokens(bigram, tweet, token = 'ngrams', n = 2)

head(tweets_bigram)
```

```
##   tweet_id       bigram
## 1       36      join us
## 2       36        us on
## 3       36    on friday
## 4       36 friday march
## 5       36    march 5th
## 6       36     5th from
```

As you can see from the dataframe above, some of the bigrams contain stop words that do not add much value. Lets remove the stop words. We will do this by first separating the bigram column into two seperate colums named 'word1' and 'word2'. We will then use two filter functions to remove the stop words.

```
tweets_bigram <- tweets_bigram %>%
  separate(bigram, c("word1", "word2"), sep = " ") %>%#separates on whitespace
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word)

head(tweets_bigram)
```

```
##   tweet_id    word1    word2
## 1       36   friday    march
## 2       36    march      5th
## 3       36        1      2pm
## 4       36    ccidm research
## 5       36 research  seminar
## 6       36  seminar   series
```

We can now count the bigrams and look at that output.

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

head(bigram_counts)
```

```
##      word1     word2 n
## 1  digital marketing 9
## 2    ccidm  research 6
## 3 research   seminar 6
## 4     data  industry 5
## 5 insights      data 5
## 6    jenny      yeon 5
```

Just like before, we can create a tf-idf with n-grams as well. Lets do that now.

```
tweets %>%
  unnest_tokens(bigram, tweet, token = 'ngrams', n = 2) %>%
  count(tweet_id, bigram) %>%
  bind_tf_idf(bigram, tweet_id, n) %>%
  group_by(tweet_id) %>%
  arrange(tweet_id, desc(tf_idf)) %>%
  head()
```

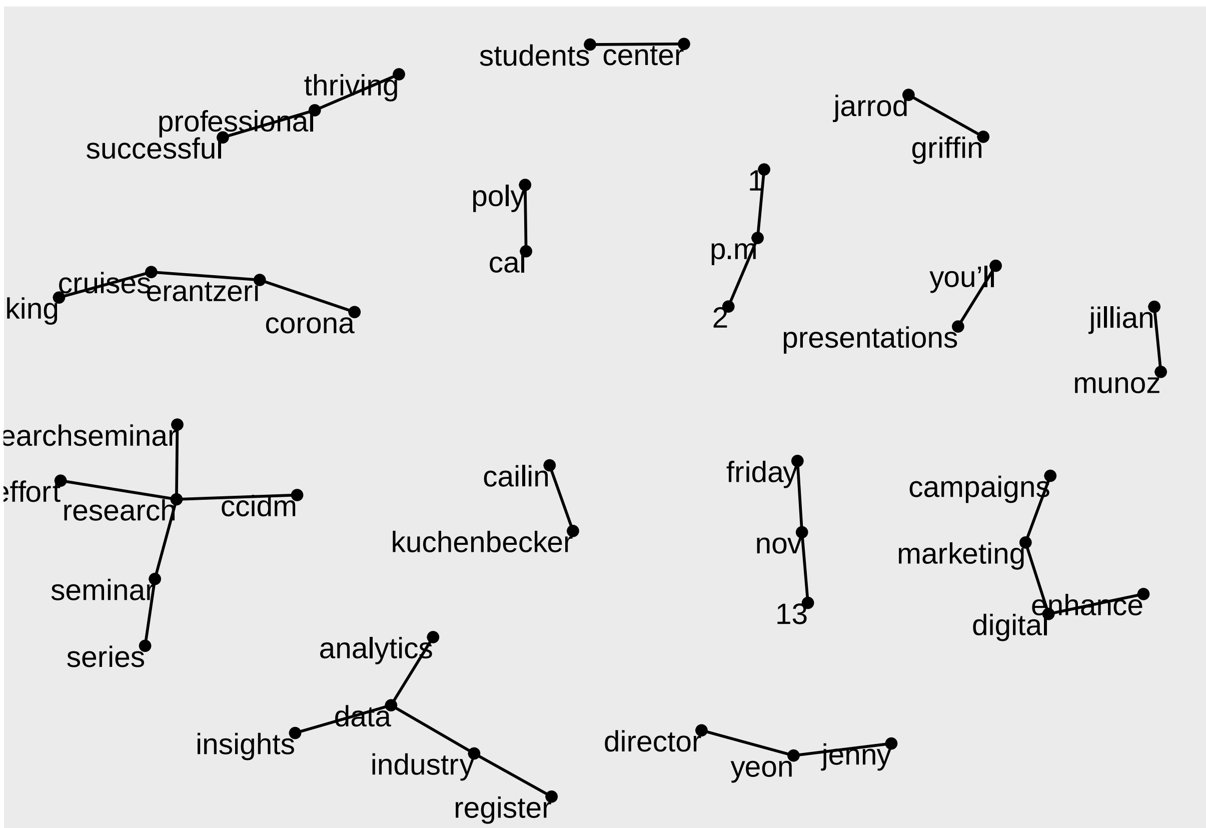
```
## # A tibble: 6 x 6
## # Groups:   tweet_id [1]
##   tweet_id bigram              n     tf   idf tf_idf
##      <int> <chr>           <int>  <dbl> <dbl>  <dbl>
## 1        1 account follow      1 0.0769  3.58  0.276
## 2        1 for regular         1 0.0769  3.58  0.276
## 3        1 new twitter         1 0.0769  3.58  0.276
## 4        1 on the              1 0.0769  3.58  0.276
## 5        1 our new             1 0.0769  3.58  0.276
## 6        1 regular updates     1 0.0769  3.58  0.276
```

As you can see from above, many of the tf-idf values are identical. This is due in part to the small sample text size of a tweet.

Lets take a visual look at word relationships between ALL of the CCIDM tweets by utilizing a network chart.

```
library('igraph')
library('ggraph')
bi_graph <- bigram_counts %>%
  filter(n > 2) %>%
  graph_from_data_frame()

ggraph(bi_graph, layout = "fr") +
  geom_edge_link() +
  geom_node_point() +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1)
```



As you can see from above, many names and other information has been mined from the CCIDM twitter data!

Tri-Grams

```
tweets_trigram <- tweets %>%
  unnest_tokens(trigram, tweet, token = 'ngrams', n = 3) %>%
  separate(trigram, c("word1", "word2", "word3"), sep = " ") %>% #separates on whitespace
  filter(!word1 %in% stop_words$word) %>%
```

```
  filter(!word2 %in% stop_words$word) %>%
  filter(!word3 %in% stop_words$word)

head(tweets_trigram)
```

```
##   tweet_id       word1       word2   word3
## 1       36      friday       march     5th
## 2       36       ccidm    research seminar
## 3       36    research     seminar  series
## 4       36    customer perceptions      dr
## 5       36 perceptions          dr    seth
## 6       36          dr        seth  ketron
```

We can now count the trigrams and look at that output.

```
trigram_counts <- tweets_trigram %>%
  count(word1, word2, word3, sort = TRUE)

head(trigram_counts)
```

```
##        word1        word2    word3 n
## 1      ccidm     research  seminar 5
## 2   insights         data industry 5
## 3       data     industry register 4
## 4      jenny         yeon director 4
## 5   research      seminar   series 4
## 6 successful professional thriving 4
```

Just like before, we can create a tf-idf with the tri-grams. Lets do that now.

```
tweets %>%
  unnest_tokens(trigram, tweet, token = 'ngrams', n = 3) %>%
  count(tweet_id, trigram) %>%
  bind_tf_idf(trigram, tweet_id, n) %>%
  group_by(tweet_id) %>%
  arrange(tweet_id, desc(tf_idf)) %>%
  head()
```

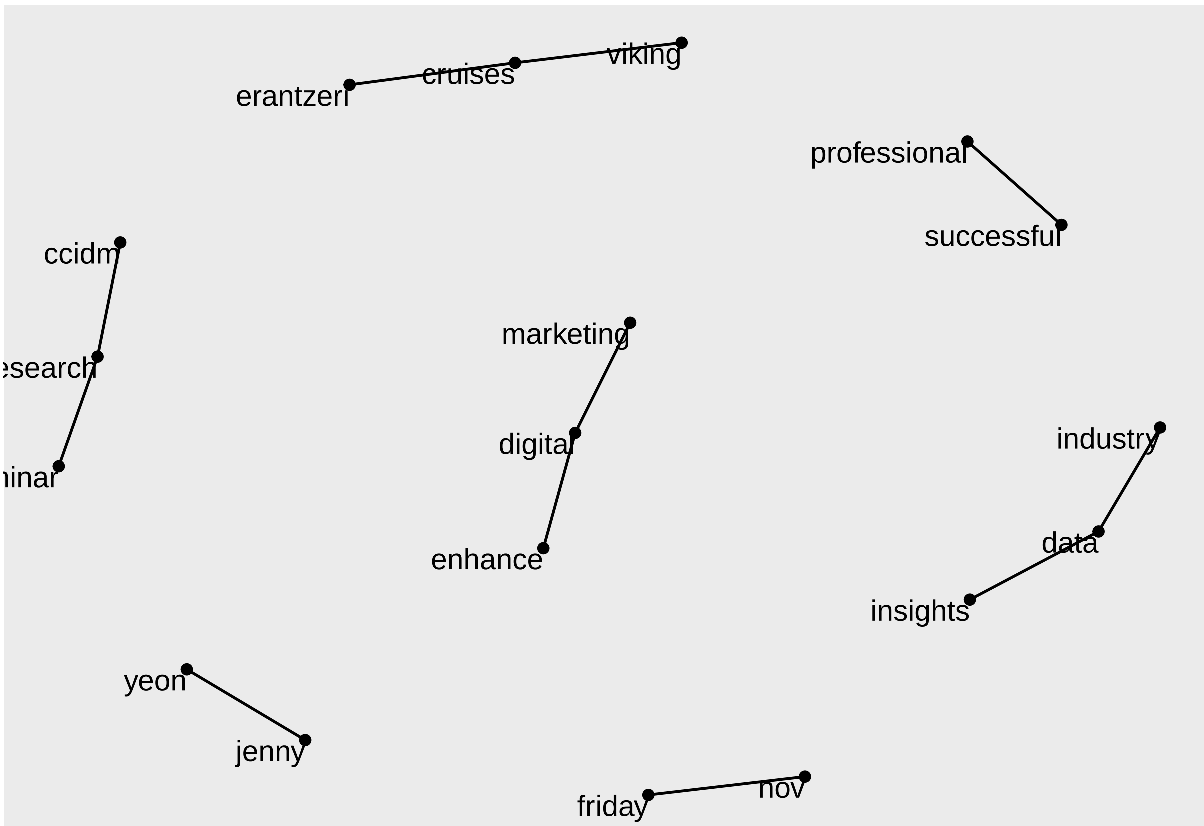
```
## # A tibble: 6 x 6
## # Groups:   tweet_id [1]
##   tweet_id trigram                  n     tf   idf tf_idf
##      <int> <chr>                <int>  <dbl> <dbl>  <dbl>
## 1        1 account follow us        1 0.0833  3.58  0.299
## 2        1 follow us for            1 0.0833  3.58  0.299
## 3        1 for regular updates      1 0.0833  3.58  0.299
## 4        1 new twitter account      1 0.0833  3.58  0.299
## 5        1 on the ccidm             1 0.0833  3.58  0.299
## 6        1 our new twitter          1 0.0833  3.58  0.299
```

As you can see from above, many of the tf-idf values are identical. This is due in part to the small sample text size of a tweet.

Lets take a visual look at word relationships between ALL of the CCIDM tweets by utilizing a network chart.

```r
library('igraph')
library('ggraph')
tri_graph <- trigram_counts %>%
  filter(n > 2) %>%
  graph_from_data_frame()

ggraph(tri_graph, layout = "fr") +
  geom_edge_link() +
  geom_node_point() +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1)
```



Topic Modeling

It is common to have a collection of documents such as news articles or social media posts that we want to divide into topics. In other words, we want to know the main topic in a document. This can be accomplished through *topic modeling*. Here, we will look at topic modeling throught the *Latent Dirichlet allocation (LDA)* method.

LDA has two main principles:

- Every document is a mixture of topics
- Every topic is a mixture of words

A common example of this is if we assume there are two main topics in news, politics and entertainment. The politics topic will have words like *elected*, or *government* whereas the entertainment topic may have words like *movie*, or *actor*. However, some words may overlap, like *award* or *budget*. LDA finds the mixture of words in each topic as well as finding the mixture of topics that describes each document. Lets demonstrate with an example below:

First we start by actually creating our LDA model. The *LDA()* function requires a DocumentTermMatrix as an input which we can create from our TF-IDF we have previously created. Below, we also use the *anti_join(stop_words)* code to remove all stop words from our TF-IDF. We can convert our TF-IDF into a DocumentTermMatrix through the *cast_dtm()* function.

```
library('topicmodels')
library('tm')

#parameters
num_topics=3
top_n_to_get=10

tweets_lda <- tweet_tf_idf %>%
  anti_join(stop_words) %>%
  cast_dtm(document = tweet_id, term =  word, value =  count) %>%
  LDA(k=num_topics)

tweets_lda
```

```
## A LDA_VEM topic model with 3 topics.
```

After we create our LDA topic model, we can use the *tidy()* function to convert the LDA into an easy to understand and use tibble. The beta column produced is the per-topic-per-word probability which is the probability of the term being generated from a topic.

```
tweet_topics <- tidy(tweets_lda) #beta is per-topic-per-word probabilities

head(tweet_topics)
```

```
## # A tibble: 6 x 3
##   topic term       beta
##   <int> <chr>     <dbl>
## 1     1 team  1.53e-198
## 2     2 team  2.04e-  2
## 3     3 team  4.00e-198
## 4     1 data  1.52e-194
## 5     2 data  2.72e-  2
## 6     3 data  3.01e-  2
```

Great, now we have a simple and easy to use tibble! We have the probability of each word appearing in each topic. Now we will work to visualize the words in each topic. It will be most helpful here to find the top 10 or so words from each topic so that we can get a better understanding of the topic. In order to do this, we need to first get the top 10 words for each topic. This can be accomplished through use of some dplyr verbs below.

```
tweet_topics_top_terms <- tweet_topics %>%
  group_by(topic) %>%
  top_n(top_n_to_get, beta) %>%
  ungroup() %>%
```

```
  arrange(topic, -beta)

head(tweet_topics_top_terms)
```

```
## # A tibble: 6 x 3
##   topic term       beta
##   <int> <chr>     <dbl>
## 1     1 research 0.0843
## 2     1 ccidm    0.0389
## 3     1 center   0.0272
## 4     1 p.m      0.0272
## 5     1 students 0.0272
## 6     1 friday   0.0272
```

Now that we have our top 10 terms per topic, we can visualize them in order to get a better grasp on what each topic is about.

```
tweet_topics_top_terms %>%
  mutate(term = reorder_within(term, beta, topic)) %>%
  ggplot(aes(beta, term, fill = factor(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ topic, scales = "free") +
  scale_y_reordered()
```