

# Text Mining in R

Jarrod Griffin

1/14/2021

## Importing Data

Tweets from the CCIDM Twitter page (<https://twitter.com/CCIDM>) were downloaded using the `snsraper` 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.

```
#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
## 2 We would like to thank everyone who participated in our workshops, attended our events, and helped
## 3
## 4
## 5           Our full, six part Virtual R Workshop Series
## 6           Join the CCIDM, AMI, and Insights Association on November 11th
## 6           Thank you to those who attended!

##   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**.

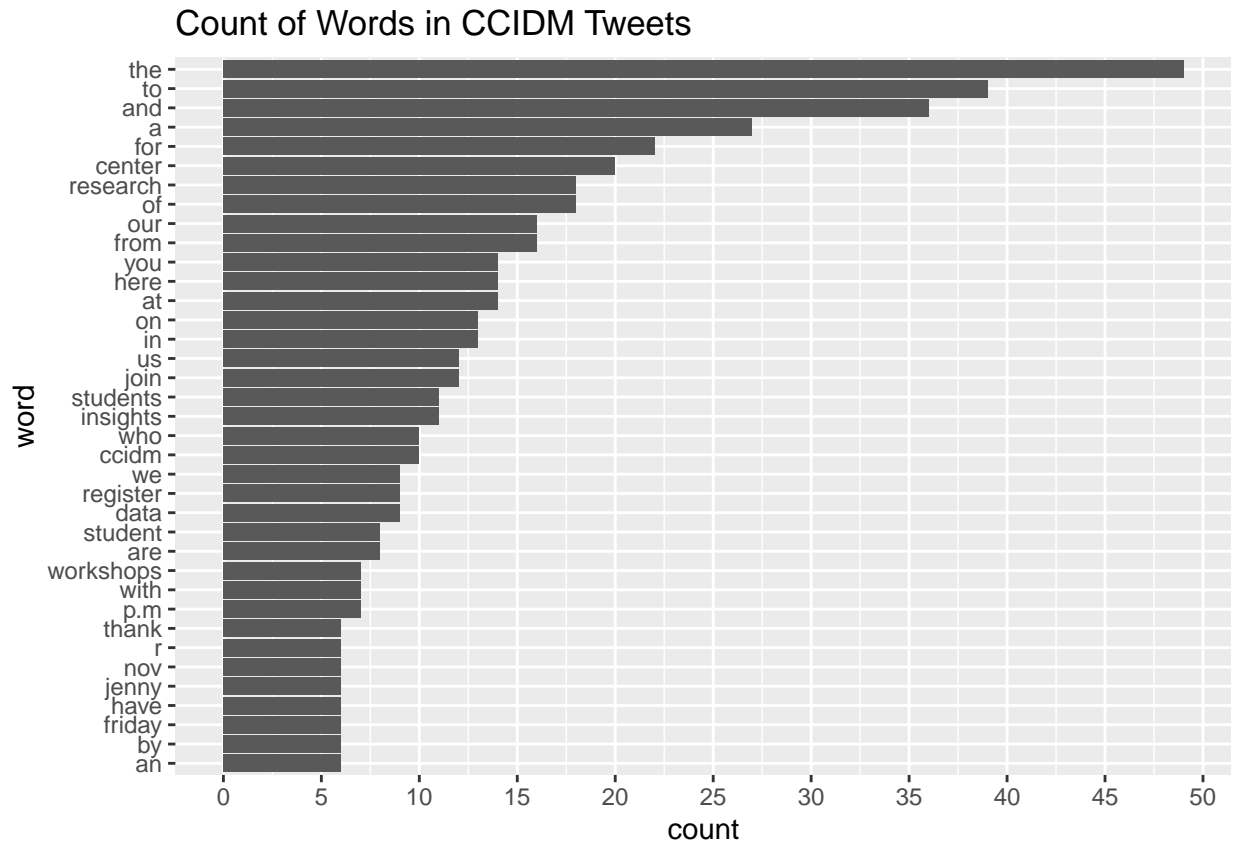
A token is defined as a **meaningful unit of text such as a word, sentence, paragraph or n-gram**.

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)
```

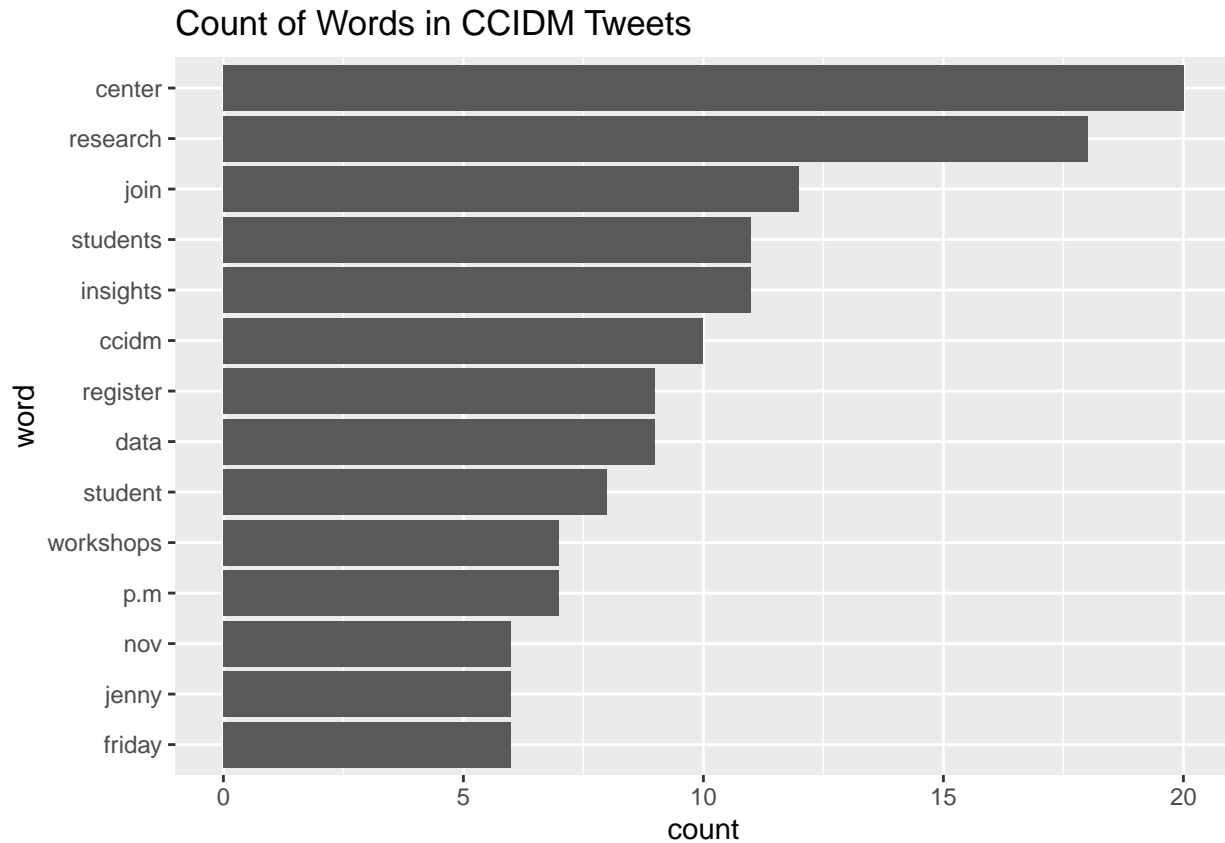
```
##   tweet_id word
## 1      31   we
## 2      31 would
## 3      31  like
## 4      31   to
## 5      31 thank
## 6      31 ccidm
```

```
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)
```

marketing director  
industry friday student successful  
team students join insights faculty  
series ccidm center data nov  
yeon register research p.m  
seminar workshops jenny jarrod  
workshop

## ##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. Let's 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
```

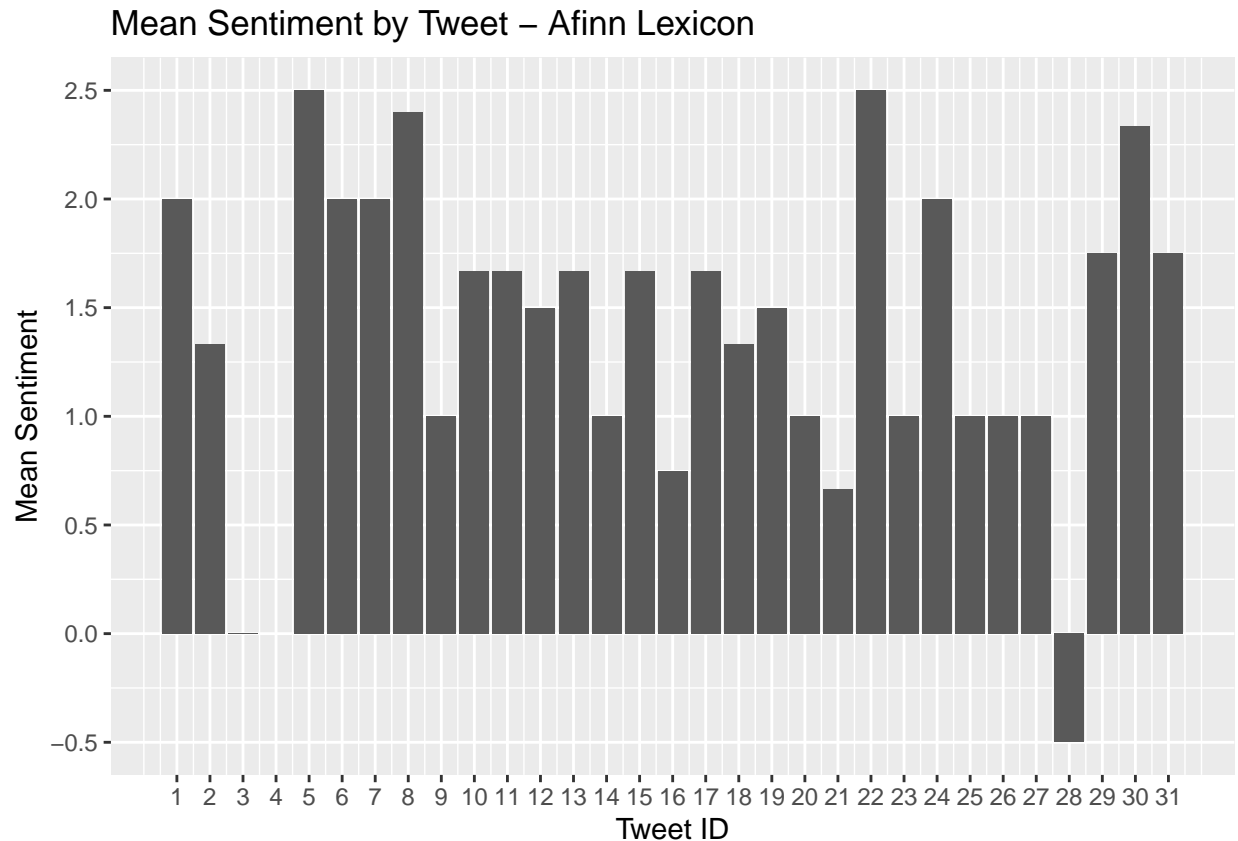
```
## # A tibble: 6,786 x 2
##   word      sentiment
##   <chr>     <chr>
## 1 2-faces    negative
## 2 abnormal  negative
## 3 abolish   negative
## 4 abominable negative
## 5 abominably negative
## 6 abominate  negative
## 7 abomination negative
## 8 abort      negative
## 9 aborted    negative
## 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 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] "The center hosts educational workshops so center members can further develop their public speak
```

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.

```
#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)
```

negative



positive

## 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 proportional 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
```

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))
```

new ccidm account follow for on	1
teams accepted team and into phase	2
campus againsta are be injustice	3
by center their and are can	4
this center the from to 2020	5

```

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))

```

updates regular account follow twitter welcome	1
accepted teams team phase into been	2
proud injustice against stands campus racial	3
educational develop co their can demonstrate	4
globe announce this korea around event	5

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 through n-grams, where n is a number.

Previously we had tokenized by single words, but we can also tokenize by n number of words. Let's 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       31    we would
## 2       31  would like
## 3       31    like to
## 4       31    to thank
## 5       31  thank ccidm
## 6       31 ccidm alumnus
```

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 separate columns 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      31      ccidm      alumnus
## 2      31  alumnus williamatienza2
## 3      31    joining      center
## 4      31 undergraduate      student
## 5      31         cal         poly
## 6      31         poly      pomona
```

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    ccidm research  5
## 2     data industry  5
## 3 insights    data  5
## 4   jenny     yeon  5
## 5 research seminar  5
## 6 industry register  4
```

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()
```

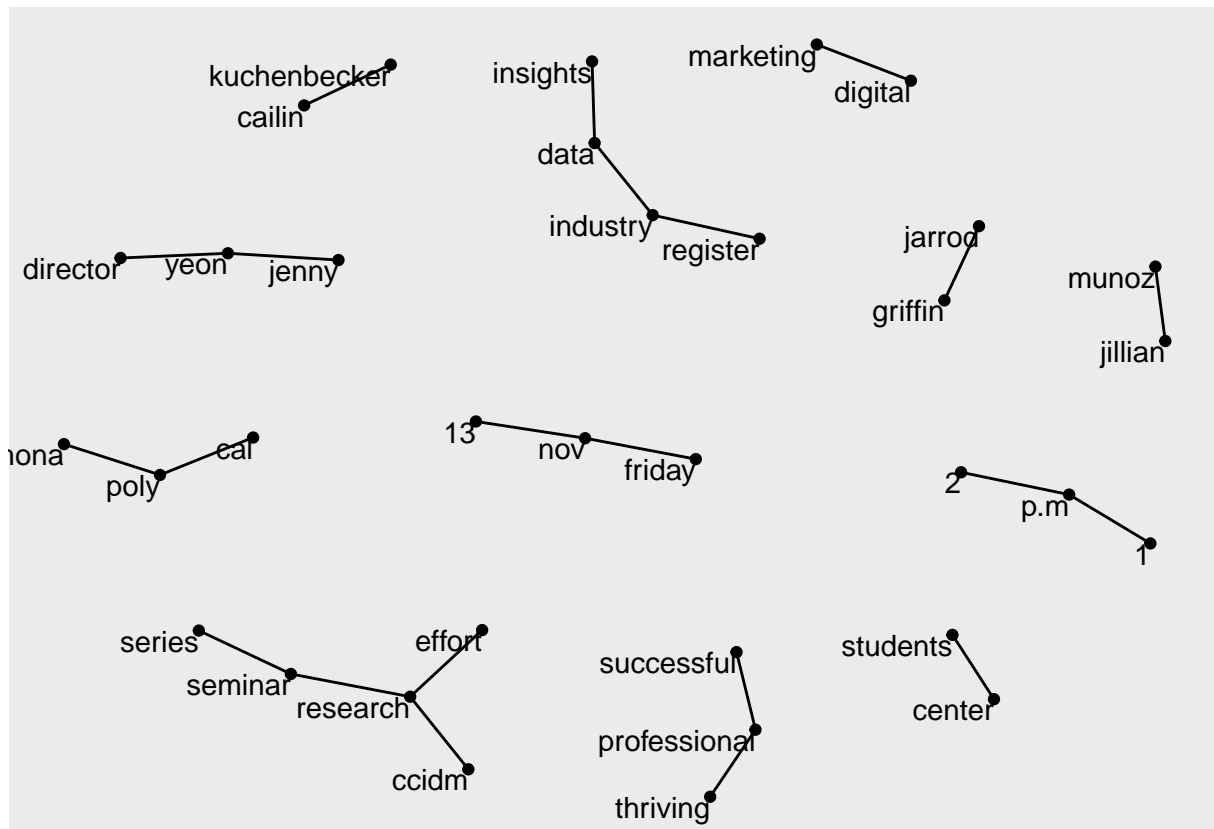
```
## # A tibble: 6 x 6
## # Groups:   tweet_id [1]
##   tweet_id bigram      n    tf  idf tf_idf
##   <int> <chr>    <int> <dbl> <dbl> <dbl>
## 1      1 account follow  1 0.0769 3.43 0.264
## 2      1 follow us      1 0.0769 3.43 0.264
## 3      1 for regular    1 0.0769 3.43 0.264
## 4      1 new twitter    1 0.0769 3.43 0.264
## 5      1 on the         1 0.0769 3.43 0.264
## 6      1 our new        1 0.0769 3.43 0.264
```

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.

```
#install.packages('igraph')
#install.packages('ggraph')
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!

## 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 through 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.

```
#install.packages('topicmodels')
#install.packages('tm')
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  1.64e-187
## 2     2 team  2.37e- 2
## 3     3 team  6.16e- 3
## 4     1 data  2.07e- 2
## 5     2 data  2.37e- 2
## 6     3 data  2.19e-175
```

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 0.0373
## 2     1  center  0.0373
## 3     1 insights 0.0290
## 4     1  join    0.0290
## 5     1 register 0.0290
## 6     1  jenny   0.0249
```

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()
```

