Exploratory Analysis

Matthew Cassi

August 30, 2016

Introduction

The purpose of this analysis is to examine data from Twitter, News sources, and Blog posts with the end goal of creating a predictive text model. In this analysis, the data, consisting of millions of lines of text, are manipulated to remove certain characters and punctuation and then transformed into sequences of words or ngrams. The model used will use these sequences of words to predict the next word in a line of text.

Basic Analysis of Data

The data set, provided by SwiftKey, contains 3 large files: 1) Tweets from Twitter, 2) Clips of News articles from various news sources, and 3) blog posts from various blogs.

```
twitter <- readLines('en_US.twitter.txt')
blogs <- readLines('en_US.blogs.txt')
news <- readLines('en_US.news.txt')
```

```
textTable <- data.frame(File = c("Twitter", "Blog", "News"),
                        Size = c(twitSize, blogSize, newsSize),
                        Lines = c(twitLines, blogLines, newsLines),
                        Words = c(twitWords, blogWords, newsWords), row.names = NULL)
kable(textTable)
```

File	Size	Lines	Words
Twitter	159.3641	2360148	30218125
Blog	200.4242	899288	38154238
News	196.2775	77259	2693898

Because these files are so large and most computers cannot handle that much data in memory, only a subset of each data set will be used in this analysis. The following R Code takes a random subset of each data set and the seed is set in order to make this analysis reproducible.

```
set.seed(2112)
smallTwitter <- sample(twitter, length(twitter)*.0025)
smallBlogs <- sample(blogs, length(blogs)*.0025)
smallNews <- sample(news, length(news)*.0025)
```

The next set of R Code combines the three data sets so that the analysis and model are based on a subset of each set.

```
smallCombo <- paste(smallBlogs, smallTwitter, smallNews, collapse = "")
```

Data Cleanup

Data cleanup is an essential piece in predictive text models. The following lines of code do a multitude of things like changing the encoding from latin to ASCII and then turn each line of code into separate sentences. Turning each line into sentences is key for the predictive word model because correct sequences of words will not be produced when punctuation is removed.

```
Encoding(smallCombo) <- "latin1"
smallCombo <- iconv(smallCombo, "latin1", "ASCII", sub="")
sent_token_annotator <- Maxent_Sent_Token_Annotator()
sent_token_annotator
```

```
## An annotator inheriting from classes
##   Simple_Sent_Token_Annotator Annotator
## with description
##   Computes sentence annotations using the Apache OpenNLP Maxent
##   sentence detector employing the default model for language 'en'.
```

```
smallCombo <- as.String(smallCombo)
a1 <- NLP::annotate(smallCombo, sent_token_annotator)
smallCombo <- smallCombo[a1]
```

The next thing to do is remove punctuation, white space, numbers, and bad words. In order to remove bad words, a list of words must be loaded, which is what the R Code below does. The bad words come from the following website: https://github.com/LDNOOBW/List-of-Dirty-Naughty-Obscene-and-Otherwise-Bad-Words/blob/master/en.

```
badWords <- readLines("./badwords.txt")
```

The next piece of R Code tokenizes the file (removes punctuation, numbers, bad words, etc.). The function tokennizeFile is a custom function that encapsulates all of the processes into one.

```
source('./tokenizeFile.R')
tokenCombo <- tokenizeFile(smallCombo, badWords)
```

After this is complete, the data are split into individual sentences (each sentence is a line), with no numbers, punction, white space, or bad words.

Exploratory Analysis

The exploratory analysis will go through the data to find the most frequent words (unigram), 2 word combinations (bigram), 3 word combinations (trigram), and 4 word combinations (quadgram).

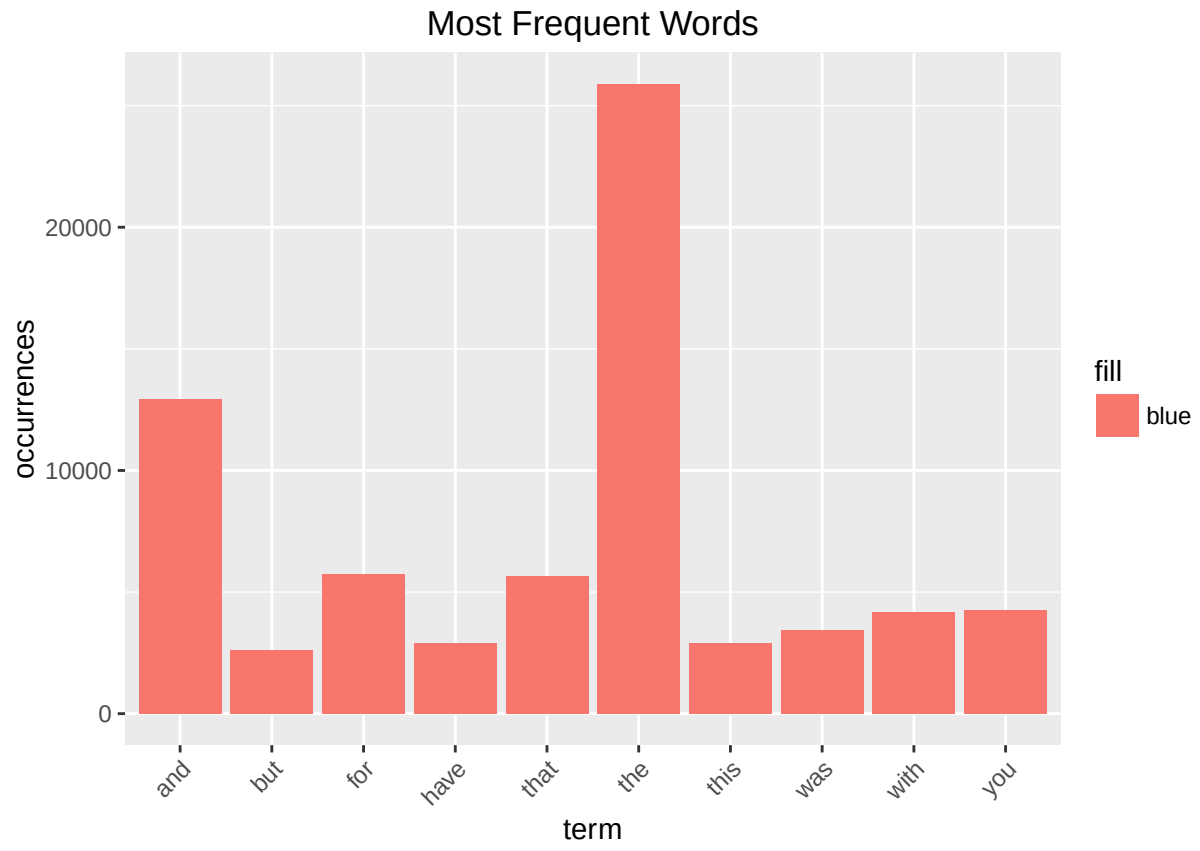
Unigram Analysis

The unigram analysis consists of making a matrix of all terms in the data as columns and each line in the data set rows. If a row consists of a word, a 1 will be placed in that "cell" of the matrix. The columns will then be summed up and arranged in order from most to least.

```
dtmCombo <- DocumentTermMatrix(tokenCombo)
freqCombo <- col_sums(dtmCombo)
```

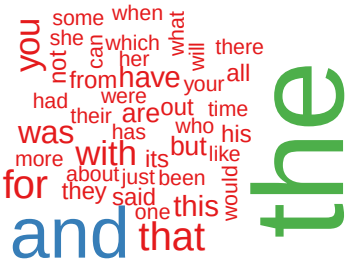
The following plot contains the most frequent terms in the data set. The frequency to appear in the bar graph must exceed 2500, that is the word must appear 2500 times in the entire data set.

```
## Creates a data frame from the list of frequent terms in the text
comboDFUnigram <- data.frame(term=names(freqCombo),occurrences=freqCombo)
## Plots the data frame using a histogram
unigram <- ggplot(subset(comboDFUnigram, freqCombo>2500), aes(term, occurrences)) +
      geom_bar(aes(fill = "blue"), stat="identity") +
      theme(axis.text.x=element_text(angle=45, hjust=1)) +
      ggtitle("Most Frequent Words")
unigram
```



The word cloud depicts the frequency of words in a different manner. Instead of a chart, it shows all of the frequent terms where the largest terms are bigger in size.

```
wordcloud(names(freqCombo),freqCombo,min.freq=1000,
          colors=brewer.pal(3,"Set1"))
```



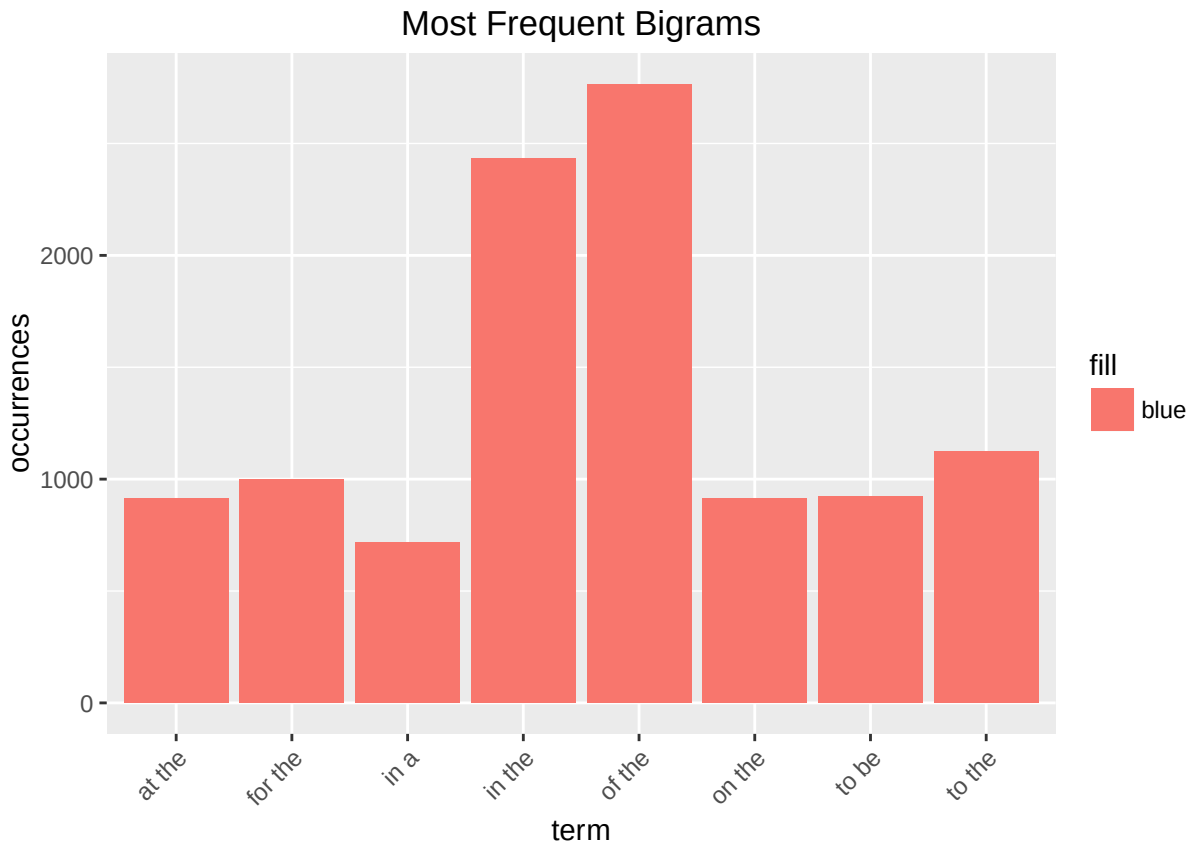
Bigram Analysis

The Bigram Analysis finds the most common two word sequences in all of the data.

```
BigramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 2, max = 2))
tdmComboBigram <- TermDocumentMatrix(Corpus(VectorSource(tokenCombo)),
                          control = list(tokenize = BigramTokenizer))
freqComboBigram <- row_sums(tdmComboBigram)
```

The following graph shows the most frequent two word sequences in the data.

```
comboDFBigram <- data.frame(term=names(freqComboBigram),occurrences=freqComboBigram, row.names = NULL)
bigram <- ggplot(subset(comboDFBigram, freqComboBigram>700), aes(term, occurrences)) +
  geom_bar(aes(fill = "blue"), stat="identity") +
  theme(axis.text.x=element_text(angle=45, hjust=1)) +
  ggtitle("Most Frequent Bigrams")
bigram
```



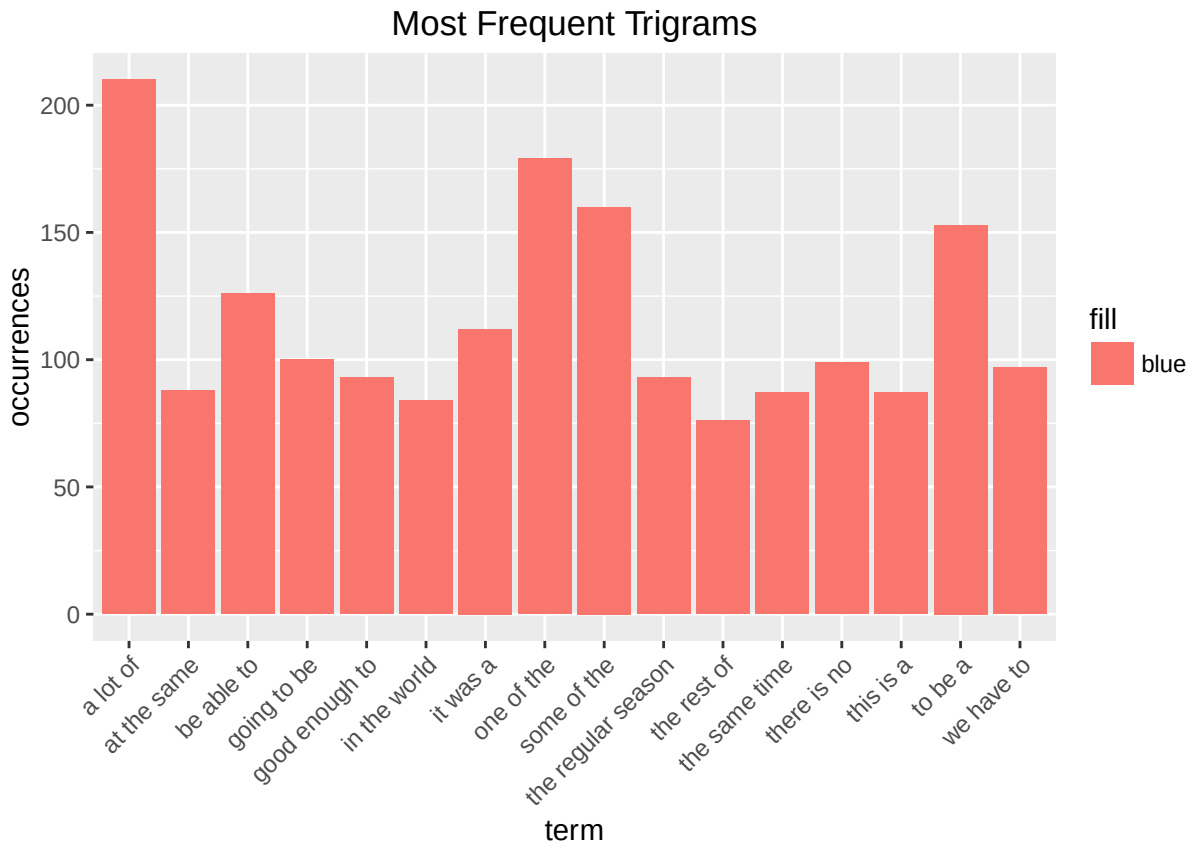
Trigram Analysis

The Trigram Analysis finds the most common three word sequences in all of the data.

```
TrigramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 3, max = 3))
tdmComboTrigram <- TermDocumentMatrix(Corpus(VectorSource(tokenCombo)),
                                      control = list(tokenize = TrigramTokenizer))
freqComboTrigram <- row_sums(tdmComboTrigram)
```

The following graph shows the most frequent three word sequences in the data.

```
comboDFTrigram <- data.frame(term=names(freqComboTrigram),occurrences=freqComboTrigram, row.names = NULL
trigram <- ggplot(subset(comboDFTrigram, freqComboTrigram>75), aes(term, occurrences)) +
  geom_bar(aes(fill = "blue"), stat="identity") +
  theme(axis.text.x=element_text(angle=45, hjust=1)) +
  ggtitle("Most Frequent Trigrams")
trigram
```



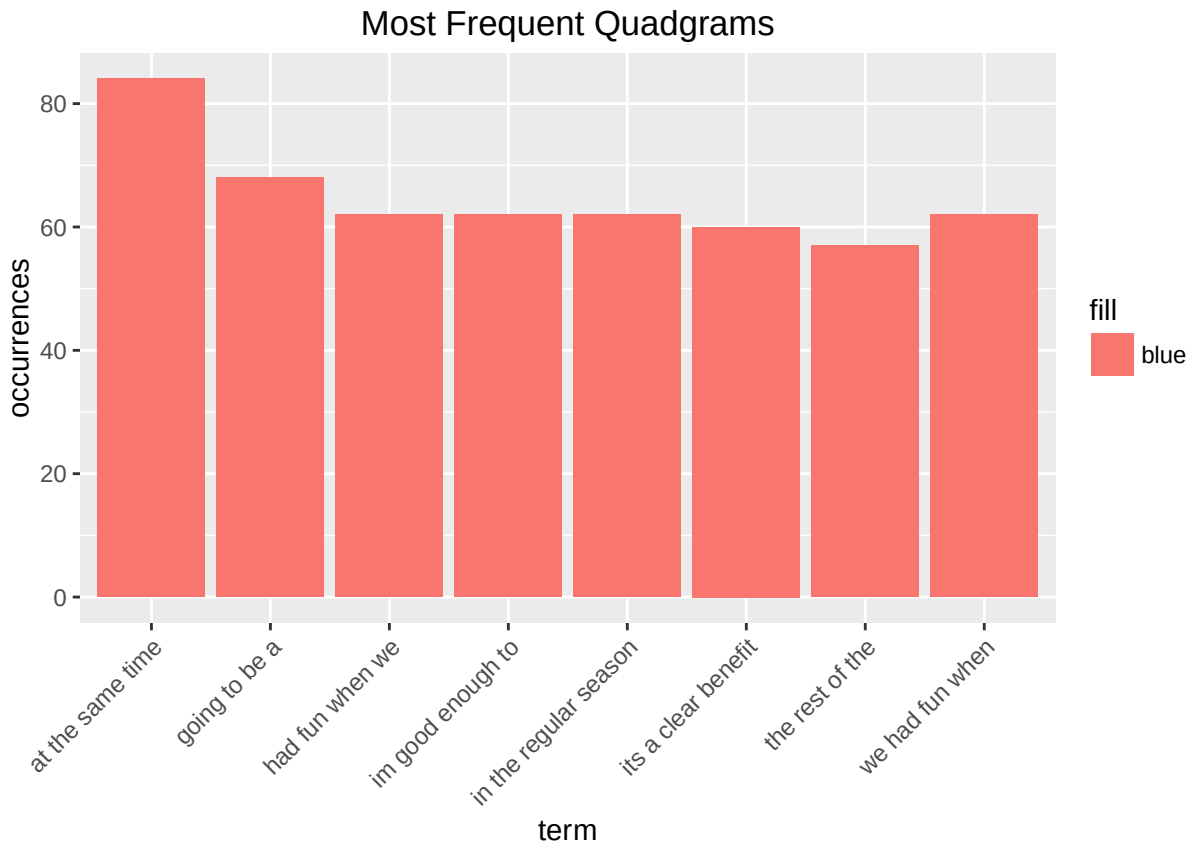
Quadgram Analysis

The Trigram Analysis finds the most common four word sequences in all of the data.

```
QuadgramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 4, max = 4))
tdmComboQuadgram <- TermDocumentMatrix(Corpus(VectorSource(tokenCombo)),
                                       control = list(tokenize = QuadgramTokenizer))
freqComboQuadgram <- row_sums(tdmComboQuadgram)
```

The following graph shows the most frequent four word sequences in the data.

```
comboDFQuadgram <- data.frame(term=names(freqComboQuadgram),occurrences=freqComboQuadgram,
                              row.names = NULL)
quadgram <- ggplot(subset(comboDFQuadgram, freqComboQuadgram>45), aes(term, occurrences)) +
  geom_bar(aes(fill = "blue"), stat="identity") +
  theme(axis.text.x=element_text(angle=45, hjust=1)) +
  ggtitle("Most Frequent Quadgrams")
quadgram
```



Next Steps for Prediction Algorithm

The following list contains the next steps for developing the predictive word model:

Take the data frames for the unigrams, bigrams, trigrams, and quadgrams and split the ngrams into the final word and remaining words. The last word and remaining words will be added as columns in each of the data frames.

Tokenize an input string that will be the basis for the prediction.

Use the tokenized string to compare this value to the different ngram data frames.

Develop a backoff algorithm to handle text that does not appear in the ngrams generated in this analysis.