Capstone Course Project - Milestone Report

Data Science Specialization from Johns Hopkins University

Prateek Sarangi

16 April, 2020

Executive Summary

This milestone report serves to show my progress with the final project for the capstone course in the data science specialization provided by Johns Hopkins University on Coursera. I will perform an exploratory data analysis to learn the structure of the data set we will be using for the final project.

Getting and Reading the Data

First we should read in the data. I use guess_encoding() here to make sure the files are read in with the proper encoding.

```
# Get directories of the data files that we want to read
us_blogs_dir <- "final/en_us/en_US.blogs.txt"
us_news_dir <- "final/en_us/en_US.news.txt"
us_twitter_dir <- "final/en_us/en_US.twitter.txt"
# Guess encoding for each file
us_blogs_encoding <- guess_encoding(us_blogs_dir, n_max=1000)$encoding[1]
us_news_encoding <- guess_encoding(us_news_dir, n_max=1000)$encoding[1]
us_twitter_encoding <- guess_encoding(us_twitter_dir, n_max=1000)$encoding[1]
# Read in files line by line
us_blogs <- readLines(us_blogs_dir, encoding=us_blogs_encoding, warn=FALSE)
us_news <- readLines(us_news_dir, encoding=us_news_encoding, warn=FALSE)
us_twitter <- readLines(us_twitter_dir, encoding=us_twitter_encoding, warn=FALSE)
```

File Statistics

As an initial exploratory measure, I'll get the sizes and number on lines in each file.

```
# calculate file sizes in MB
blogs_file_size <- file.info(us_blogs_dir)$size/(1024^2)
news_file_size <- file.info(us_news_dir)$size/(1024^2)
twitter_file_size <- file.info(us_twitter_dir)$size/(1024^2)
# Combine file sizes
file_sizes <- rbind(blogs_file_size, news_file_size, twitter_file_size)
# Count number of lines in each file
blogs_file_lines <- countLines(us_blogs_dir)
news_file_lines <- countLines(us_news_dir)
twitter_file_lines <- countLines(us_twitter_dir)
# Combine number of lines
num_lines <- rbind(blogs_file_lines, news_file_lines, twitter_file_lines)
```

```
# Combine file encodings
encodings <- rbind(us_blogs_encoding, us_news_encoding, us_twitter_encoding)
# Combine both stats
file_stats <- as.data.frame(cbind(file_sizes, num_lines, encodings))
colnames(file_stats) <- c("File Size (in MB)","Number of Lines", "File Encoding")
rownames(file_stats) <- c("Blogs", "News", "Twitter")
file_stats
```

```
##         File Size (in MB) Number of Lines File Encoding
## Blogs    200.424207687378          899288         UTF-8
## News     196.277512550354         1010242         UTF-8
## Twitter  159.364068984985         2360148         UTF-8
```

Sampling

These data sets are very large so we have to take samples from them to make the data managable.

```
# Set seed
set.seed(12345)
# Grab samples from raw data
blogs_sample <- sample(us_blogs, size=10000)
news_sample <- sample(us_news, size=10000)
twitter_sample <- sample(us_twitter, size=10000)
```

Corpus

Now we can combine the samples into a single text corpus

```
# Combine sample sets to create corpus for training
corpus_raw <- c(blogs_sample, news_sample, twitter_sample)
```

Memory Usage

The raw data from the previous steps takes up quite a bit of memory so let's remove them to free up some space.

```
# Remove raw-er data sets
rm(us_blogs, blogs_sample,
   us_news, news_sample,
   us_twitter, twitter_sample)
```

Cleaning

Now we can remove some unwanted words and punctutation characters. Let's make a function that will make things easier.

```
# changes special characters to a space character
change_to_space <- content_transformer(function(x, pattern) gsub(pattern, " ", x))
```

Begin cleaning!

```
# Remove non-ASCII characters
corpus <- iconv(corpus_raw, "UTF-8", "ASCII", sub="")
# Make corpus
corpus <- VCorpus(VectorSource(corpus))
```

```
## Begin cleaning
# Lowercase all characters
corpus <- tm_map(corpus, content_transformer(tolower))
# Strip whitespace
corpus <- tm_map(corpus, stripWhitespace)
# Remove numbers
corpus <- tm_map(corpus, removeNumbers)
# Remove punctuation characters
corpus <- tm_map(corpus, removePunctuation)
# Remove other characters
corpus <- tm_map(corpus, change_to_space, "/|@|\\|")
# Remove stop words
corpus <- tm_map(corpus, removeWords, stopwords("english"))
```

Tokenization

Now we can create our N-gram models. For our purposes we will only go up to trigrams.

```
delims <- " \\r\\n\\t.,;:\"()?!"
tokenize_uni <- function(x){NGramTokenizer(x, Weka_control(min=1, max=1, delimiters=delims))}
tokenize_bi  <- function(x){NGramTokenizer(x, Weka_control(min=2, max=2, delimiters=delims))}
tokenize_tri <- function(x){NGramTokenizer(x, Weka_control(min=3, max=3, delimiters=delims))}
unigram <- TermDocumentMatrix(corpus, control=list(tokenize=tokenize_uni))
bigram  <- TermDocumentMatrix(corpus, control=list(tokenize=tokenize_bi ))
trigram <- TermDocumentMatrix(corpus, control=list(tokenize=tokenize_tri))
```

Exploratory Data Analysis

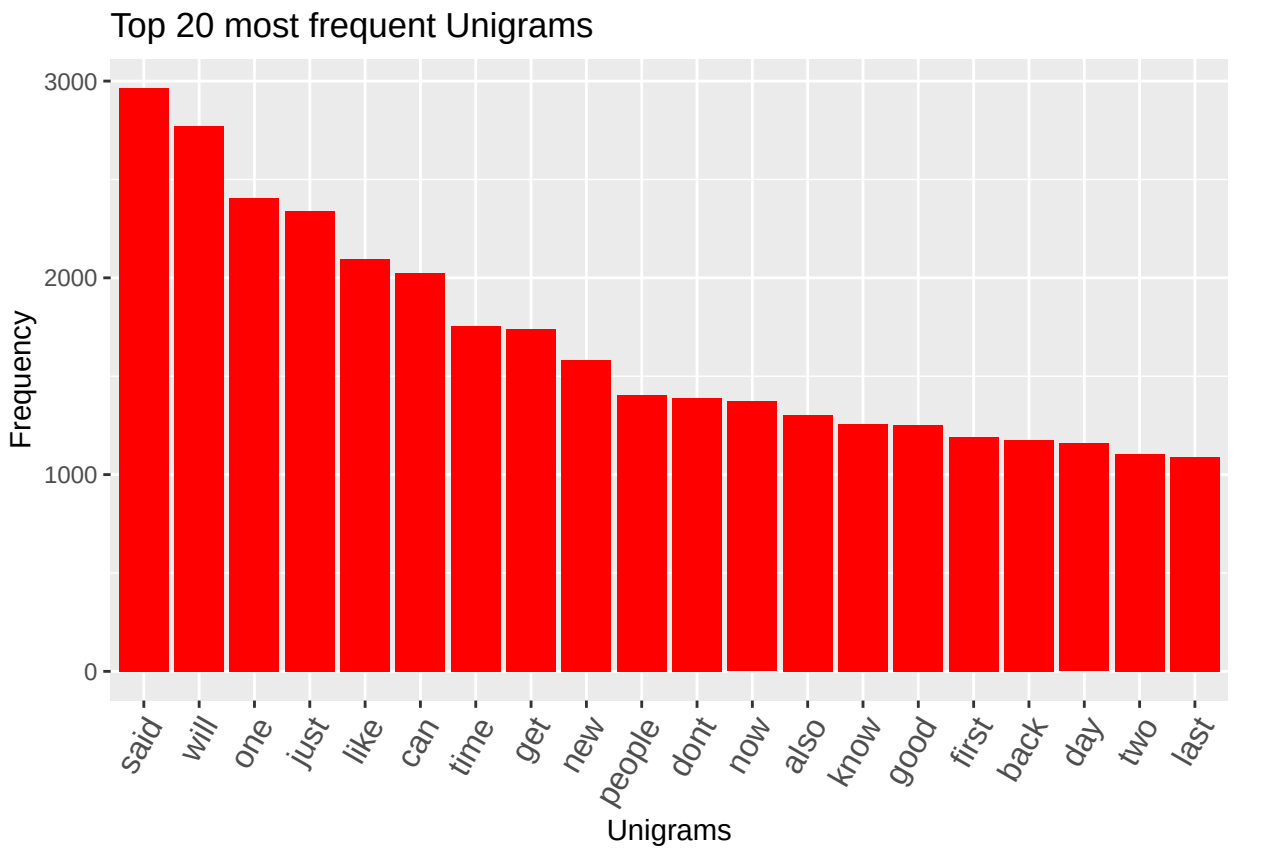
Let's count up our most frequent tokens from each N-gram.

```
# Transform N-grams structure to pull out token frequencies
unigram_r <- rollup(unigram, 2, na.rm = TRUE, FUN = sum)
 bigram_r <- rollup( bigram, 2, na.rm = TRUE, FUN = sum)
trigram_r <- rollup(trigram, 2, na.rm = TRUE, FUN = sum)
# Get token frequencies of each N-gram
unigram_tokens_counts <- data.frame(Token = unigram$dimnames$Terms, Frequency = unigram_r$v)
 bigram_tokens_counts <- data.frame(Token =  bigram$dimnames$Terms, Frequency =  bigram_r$v)
trigram_tokens_counts <- data.frame(Token = trigram$dimnames$Terms, Frequency = trigram_r$v)
# Sort tokens by frequency
unigram_sorted <- arrange(unigram_tokens_counts, desc(Frequency))
 bigram_sorted <- arrange( bigram_tokens_counts, desc(Frequency))
trigram_sorted <- arrange(trigram_tokens_counts, desc(Frequency))
# Save sorted data for word prediction later
save(unigram_sorted, file = "ngrams/unigram.RData")
save( bigram_sorted, file = "ngrams/bigram.RData" )
save(trigram_sorted, file = "ngrams/trigram.RData")
# Filter top 100 most frequent
top_unigram <- top_n(unigram_sorted, 100, Frequency)
 top_bigram <- top_n( bigram_sorted, 100, Frequency)
top_trigram <- top_n(trigram_sorted, 100, Frequency)
```

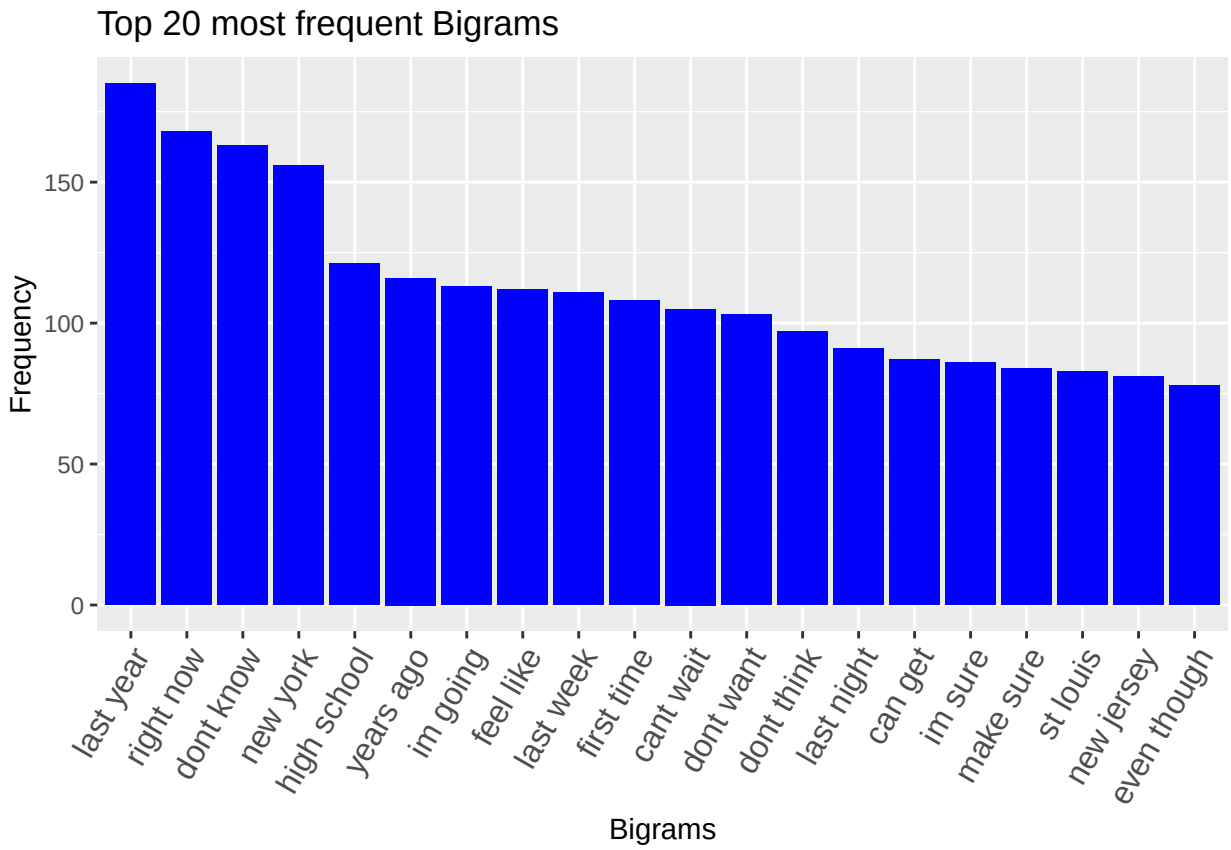
Barplots of Token Frequency

Finally, we can make some barplots showing the frequencies of the most common tokens in each N-gram.

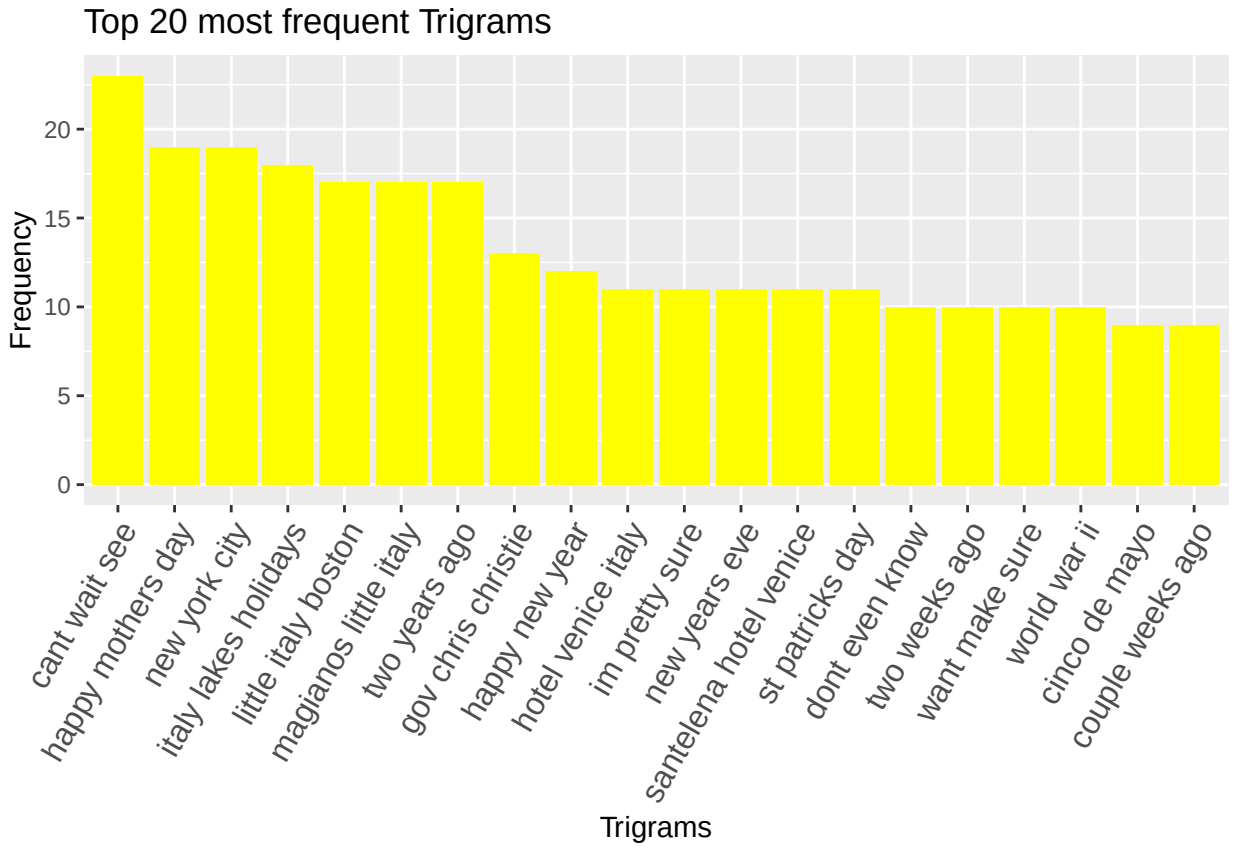
```
make_ngram_barplot <- function(x, top_n, n, color){
  main_title <- paste("Top", as.character(top_n), "most frequent", n)
  ggplot(x[1:top_n,], aes(reorder(Token, -Frequency), Frequency)) +
    geom_bar(stat="identity", fill=I(color)) +
    labs(x=n, y="Frequency") + ggtitle(main_title) +
    theme(axis.text.x = element_text(angle = 60, size = 12, hjust = 1))
}
make_ngram_barplot(top_unigram, 20, "Unigrams", "red")
```




```
make_ngram_barplot(top_bigram,  20,  "Bigrams", "blue")
```



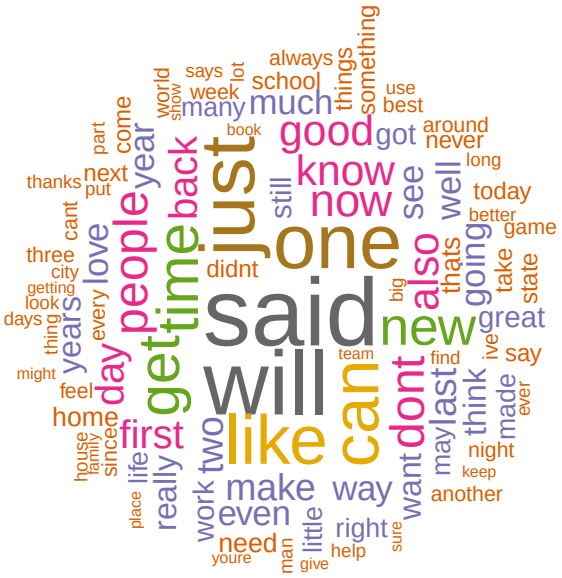

```
make_ngram_barplot(top_trigram, 20, "Trigrams", "yellow")
```



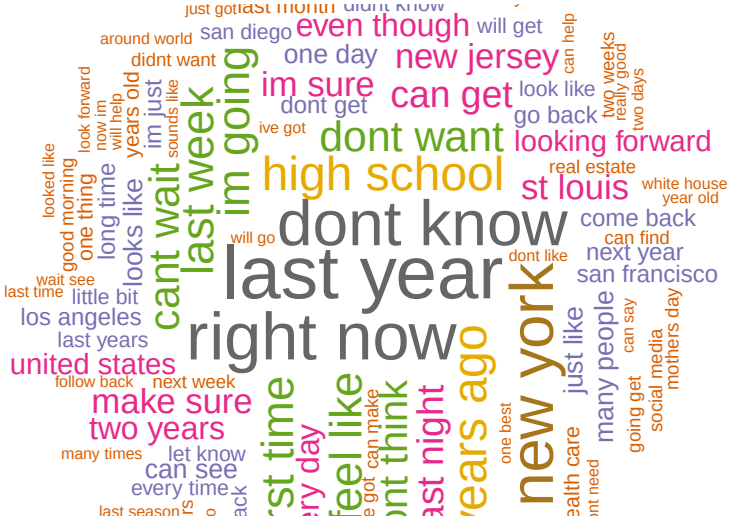
Word Clouds

We can also make word clouds as another way to visualize our token frequencies.

```
make_word_cloud <- function(x, s, max_words){
  wordcloud(x[,1], x[,2], scale=s,
          min.freq=5, max.words=max_words, random.order=FALSE,
          rot.per=0.5, colors=brewer.pal(8, "Dark2"),
          use.r.layout=FALSE)
}
make_word_cloud(top_unigram, c(3.0, 0.1), 100)
```




```
make_word_cloud(top_bigram,  c(2.3, 0.1), 100)
```




```
make_word_cloud(top_trigram, c(1.5, 0.1), 100)
```

Conclusion and Planning for Final Project

It seems that we will have to sacrifice a significant portion of our model's accuracy for the sake of runtime. Even though the sample size is pretty small compared to the raw data set, it still takes quite a while to construct our ngrams data. I think a sample size of 30,000 will be sufficient. I think a better job could be done with the data cleaning, as some of the trigrams seem to be a bit odd.

For the final project I plan on training a model using the N-grams constructed here and deploying it into a shiny application that will predict the next word from a user's input.