Milestone Report for Data Science Capstone Project

freestander

Saturday, December 28, 2015

Introduction

The goal of the project is to build a model to predict the next words based on previous input from the user. The data used to produce the predictive model is obtained from the Coursera website: https://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-SwiftKey.zip. The data contains input text files from three sources: blogs, news and twitter.

This milestone report includes a description of setting up the environment, loading the data, preprocessing the data, exploratory data analysis, findings and conclusions, and future steps. It is the foundation for building the predictive model and the Shinny App in the next phase.

Setting up the Environment

We need to set up the working directory and load the necessary packages used in the analysis.

```r
# set the working directory
setwd("C:/Users/Qi-Desktop/Documents/RProjects/Coursera/Data_Science_Capstone")

# text mining
library(tm)
# N-GramS vector generation
library(RWeka)
# plotting figures
library(ggplot2)
```

Loading the Data

First, we briefly check the size for the three files: blogs, news, and tweets (in Mb).

```r
# set the working directory
setwd("C:/Users/Qi-Desktop/Documents/RProjects/Coursera/Data_Science_Capstone")

# size of the blogs file
blogs_size <- file.info("./data/en_US.blogs.txt")$size/1024^2
# size of the news file
news_size <- file.info("./data/en_US.news.txt")$size/1024^2
# size of the twitter file
twitter_size <- file.info("./data/en_US.twitter.txt")$size/1024^2
```

Second, we load the three files into R console. We also use a list of profane words downloaded from http://www.cs.cmu.edu/~biglou/resources/bad-words.txt.

```r
# set the working directory
setwd("C:/Users/Qi-Desktop/Documents/RProjects/Coursera/Data_Science_Capstone")
```

```r
# read blogs data
con <- file("./data/en_US.blogs.txt", "r")
blogs <- readLines(con)
close(con)

# read news data
con <- file("./data/en_US.news.txt", "r")
news <- readLines(con)
close(con)

# read twitter data
con <- file("./data/en_US.twitter.txt", "r")
twitter <- readLines(con)
close(con)

# read profane words
con <- file("./data/bad_words.txt", "r")
bad_words <- readLines(con)
close(con)
```

Third, we check the line counts for the three files.

```r
blogs_lines <- length(blogs)
news_lines <- length(news)
twitter_lines <- length(twitter)
```

Fourth, we check the number of words in the three files.

```r
# check # of words in blogs
blogs_words <- sum(sapply(gregexpr("\\s+", blogs), length) + 1)
# check # of words in news
news_words <- sum(sapply(gregexpr("\\s+", news), length) + 1)
# check # of words in twitter
twitter_words <- sum(sapply(gregexpr("\\s+", twitter), length) + 1)
```

Fifth, we check the longest line in the three files.

```r
# check maximum line in blogs
blogs_max_line <- max(nchar(blogs))
# check maximum line in news
news_max_line <- max(nchar(news))
# check maximum line in twitter
twitter_max_line <- max(nchar(twitter))
```

Finally, we gather all the information about the three files and summarize it in a table as shown below.

```r
size_table <- data.frame(cbind(c(blogs_size, news_size, twitter_size),
                               c(blogs_lines, news_lines, twitter_lines),
                               c(blogs_words, news_words, twitter_words),
                               c(blogs_max_line, news_max_line, twitter_max_line)))
rownames(size_table) <- c("blogs", "news", "twitter")
colnames(size_table) <- c("size (Mb)", "lines", "words", "max_line")

size_table
```

```
##          size (Mb)   lines    words max_line
## blogs     200.4242  899288 37345664    40835
## news      196.2775   77259  2644243     5760
## twitter   159.3641 2360148 30375242      213
```

Preprocessing the Data

Due to the large size of the files, we take 1% sample out of the data for exploratory analysis. A representative sample can be used to infer facts about a population.

```
set.seed(123)
blogs_sample <- sample(blogs, size=round(length(blogs)*0.01))
news_sample <- sample(news, size=round(length(news)*0.01))
twitter_sample <- sample(twitter, size=round(length(twitter)*0.01))
```

We then write the sample dataset out to files so that we don't need to recreate it for repeating analysis.

```
# set the working directory
setwd("C:/Users/Qi-Desktop/Documents/RProjects/Coursera/Data_Science_Capstone")

write.csv(blogs_sample, file="./data/en_US.blog_sample.csv")
write.csv(news_sample, file="./data/en_US.news_sample.csv")
write.csv(twitter_sample, file="./data/en_US.twitter_sample.csv")
```

We can merge the three sample files into one bigger file for analysis.

```
data_merged <- c(blogs_sample, news_sample, twitter_sample)
```

We write a function to clean up the undesired characters from the dataset before converting it into the corpus for data mining purposes.

```
cleanUp <- function(data) {
    data <- gsub("/|@|\\|", " ", data, perl = TRUE)
    data <- gsub("\xe2\x80\x99", "'", data, perl=TRUE)
    data <- gsub("\u000A|\u0027|\u0060|\u0091|\u0092|\u0093|\u0094|\u2019", "'", data, perl=TRUE)
    data <- iconv(data, "UTF-8", "ASCII", "?")
    data <- gsub("[^[:alpha:][:space:]'-]", " ", data)
    data <- gsub("(?<!\\w)[-'](?<!\\w)" , " ", data, perl=TRUE)
    data <- gsub("[[:space:]]+", " ", data, perl=TRUE)
    data <- gsub("^[[:space:]]+", "", data, perl=TRUE)
    data <- tolower(data)
    return(data)
    }
```

Then we call the function to clean up the sample files.

```
data_merged <- cleanUp(data_merged)
```

After that we create corpus data using tm package.

```
data_corpus <- Corpus(VectorSource(data_merged))
```

We write a function to do some further processing and remove the profane words using tm package.

```
tm_processing <- function(data, profane) {
  data <- tm_map(data, content_transformer(tolower))
  data <- tm_map(data, removeNumbers)
  data <- tm_map(data, removePunctuation)
  data <- tm_map(data, removeWords, profane)
  data <- tm_map(data, stemDocument, language = "english")
  data <- tm_map(data, stripWhitespace)
  return(data)
  }
```

Next we call the function to do the processing.

```
data_corpus <- tm_processing(data_corpus, bad_words)
```

We generate N-Grams tokenizers using RWeka package.

```
bi_tokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 2, max = 2))
tri_tokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 3, max = 3))
quad_tokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 4, max = 4))
```

Then we generate the N-Grams vector using the tokenizers. To avoid excessive memmory consumption and maintain model performance, we remove words that have sparsity more than 99.9% for Uni-Gram and Bi-Grams, 99.95% for Tri-Grams, and 99.99% for Quad-Grams. This leaves about 1738 Uni-Gram, 1638 Bi-Grams, 669 Tri-Grams, and 1313 Quad-Grams.

```
# Uni-Gram
data_uni_gram <- DocumentTermMatrix(data_corpus)
data_uni_gram_nsp <- removeSparseTerms(data_uni_gram, 0.999)

# Bi-Grams
data_bi_gram <- DocumentTermMatrix(data_corpus, control = list(tokenize = bi_tokenizer))
data_bi_gram_nsp <- removeSparseTerms(data_bi_gram, 0.999)

# Tri-Grams
data_tri_gram <- DocumentTermMatrix(data_corpus, control = list(tokenize = tri_tokenizer))
data_tri_gram_nsp <- removeSparseTerms(data_tri_gram, 0.9995)

# Quad-Grams
data_quad_gram <- DocumentTermMatrix(data_corpus, control = list(tokenize = quad_tokenizer))
data_quad_gram_nsp <- removeSparseTerms(data_quad_gram, 0.9999)
```

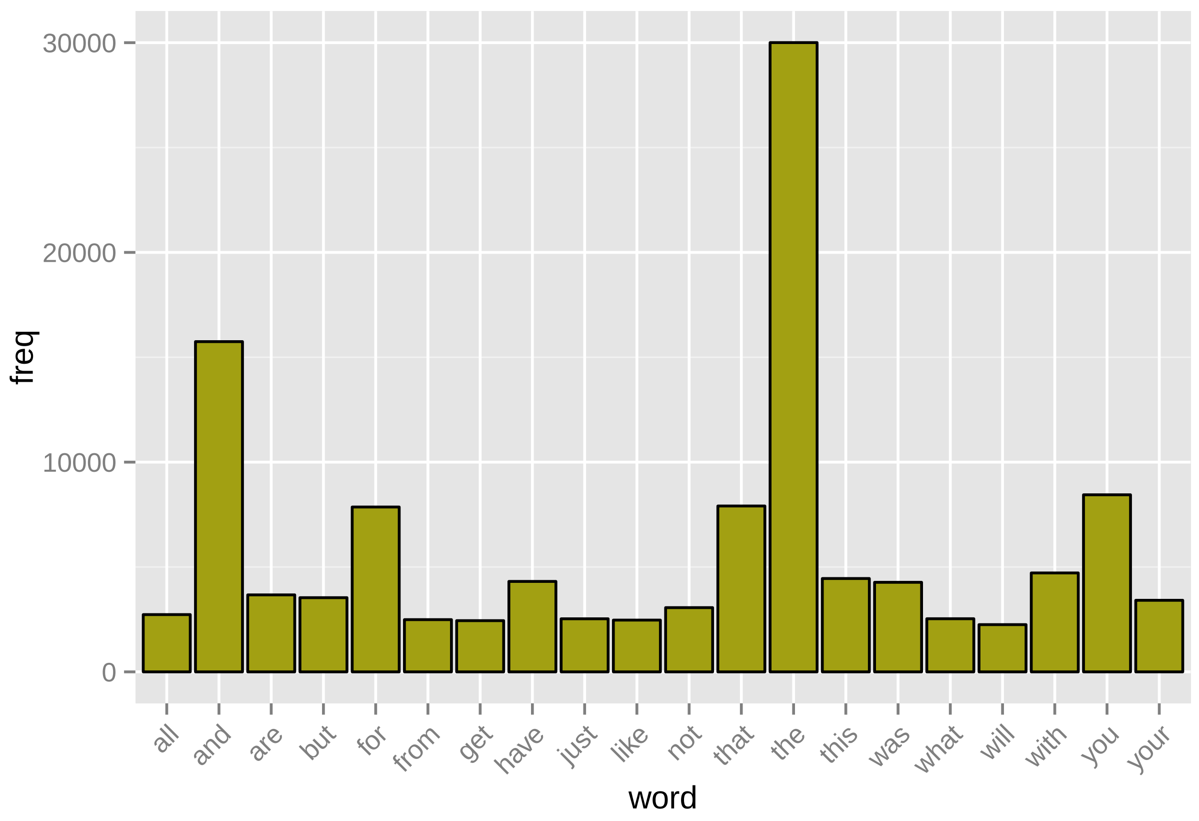
Exploratory Data Analysis

We convert the document term matrix for Uni-Gram into a word frequency table.

```
data_uni_matrix  <- as.matrix(data_uni_gram_nsp)
data_uni_rank <- sort(colSums(data_uni_matrix), decreasing=TRUE)
data_uni_rank <- data.frame(word=names(data_uni_rank), freq=data_uni_rank)
```

Show the frequency plot for the top 20 Uni-Gram.

```
library("ggplot2")
p <- ggplot(head(data_uni_rank, 20), aes(x=word, y=freq))
p <- p + geom_bar(stat="identity", fill="#A2A012", colour="black")
p <- p + theme(axis.text.x=element_text(angle=45, hjust=1))
p
```

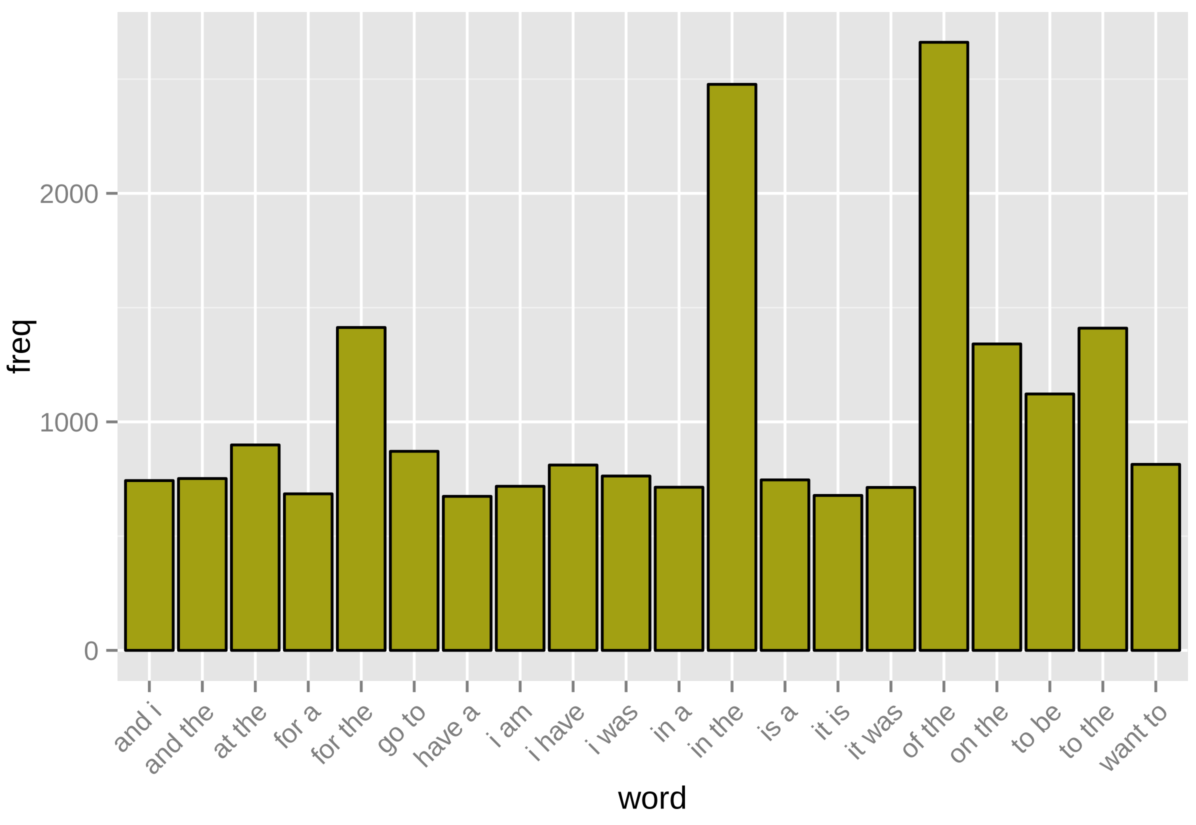


We convert the document term matrix for Bi-Grams into a word frequency table.

```
data_bi_matrix  <- as.matrix(data_bi_gram_nsp)
data_bi_rank <- sort(colSums(data_bi_matrix), decreasing=TRUE)
data_bi_rank <- data.frame(word=names(data_bi_rank), freq=data_bi_rank)
```

Show the frequency plot for the top 20 Bi-Grams.

```
library("ggplot2")
p <- ggplot(head(data_bi_rank, 20), aes(x=word, y=freq))
p <- p + geom_bar(stat="identity", fill="#A2A012", colour="black")
p <- p + theme(axis.text.x=element_text(angle=45, hjust=1))
p
```

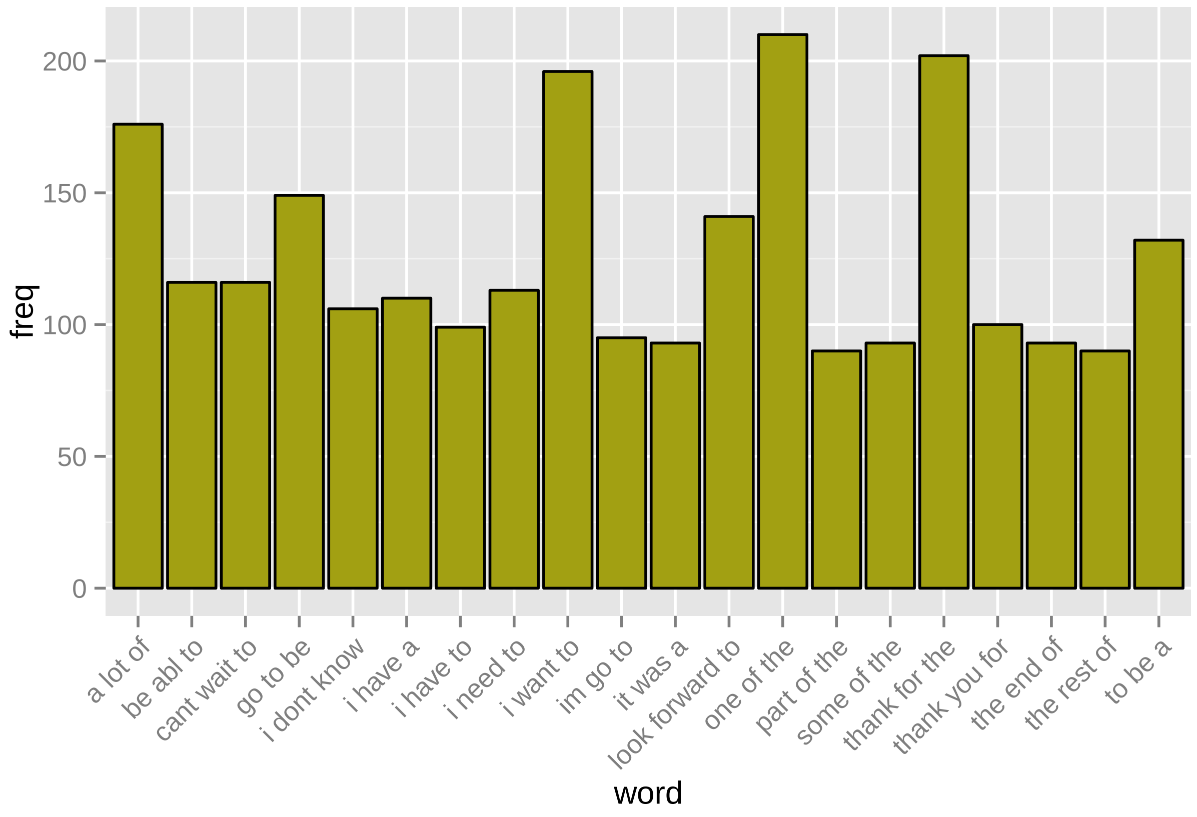


We convert the document term matrix for Tri-Grams into a word frequency table.

```
data_tri_matrix  <- as.matrix(data_tri_gram_nsp)
data_tri_rank <- sort(colSums(data_tri_matrix), decreasing=TRUE)
data_tri_rank <- data.frame(word=names(data_tri_rank), freq=data_tri_rank)
```

Show the frequency plot for the top 20 Tri-Grams.

```
library("ggplot2")
p <- ggplot(head(data_tri_rank, 20), aes(x=word, y=freq))
p <- p + geom_bar(stat="identity", fill="#A2A012", colour="black")
p <- p + theme(axis.text.x=element_text(angle=45, hjust=1))
p
```

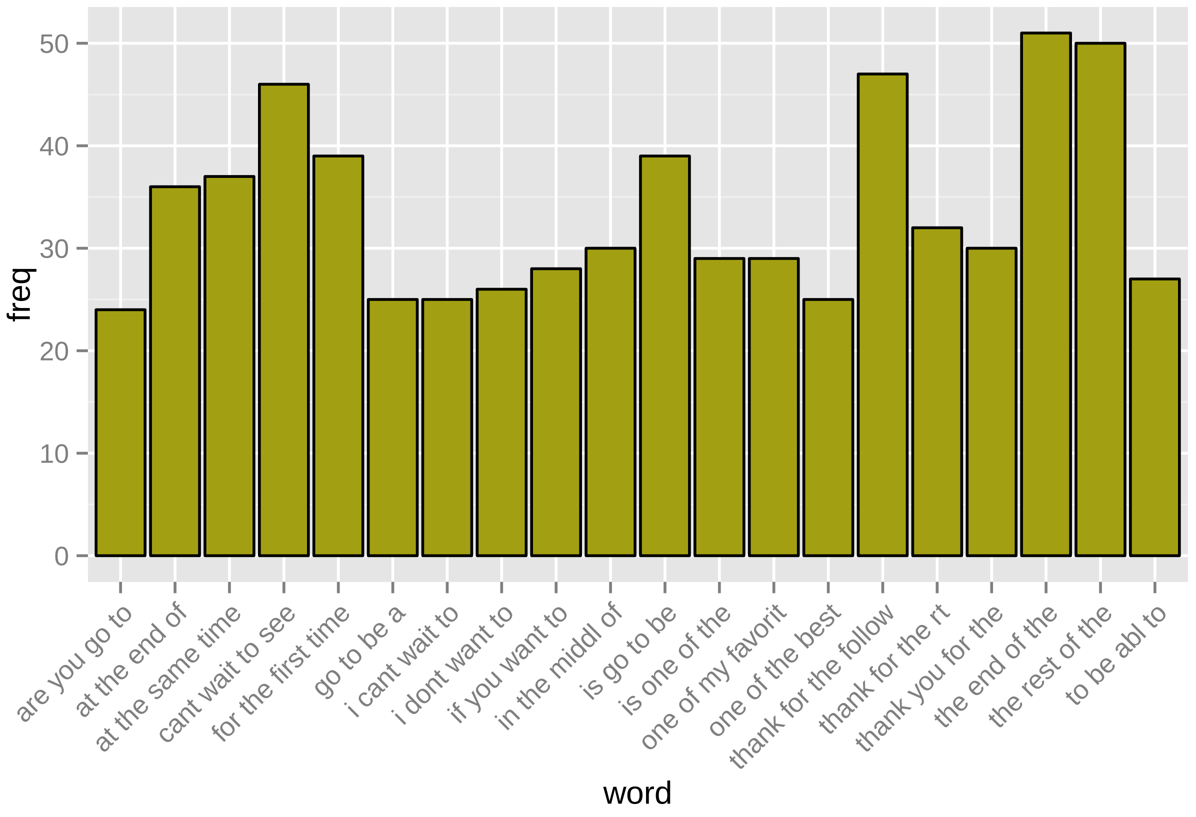


We convert the document term matrix for Quad-Grams into a word frequency table.

```
data_quad_matrix  <- as.matrix(data_quad_gram_nsp)
data_quad_rank <- sort(colSums(data_quad_matrix), decreasing=TRUE)
data_quad_rank <- data.frame(word=names(data_quad_rank), freq=data_quad_rank)
```

Show the frequency plot for the top 20 Quad-Grams.

```
library("ggplot2")
p <- ggplot(head(data_quad_rank, 20), aes(x=word, y=freq))
p <- p + geom_bar(stat="identity", fill="#A2A012", colour="black")
p <- p + theme(axis.text.x=element_text(angle=45, hjust=1))
p
```



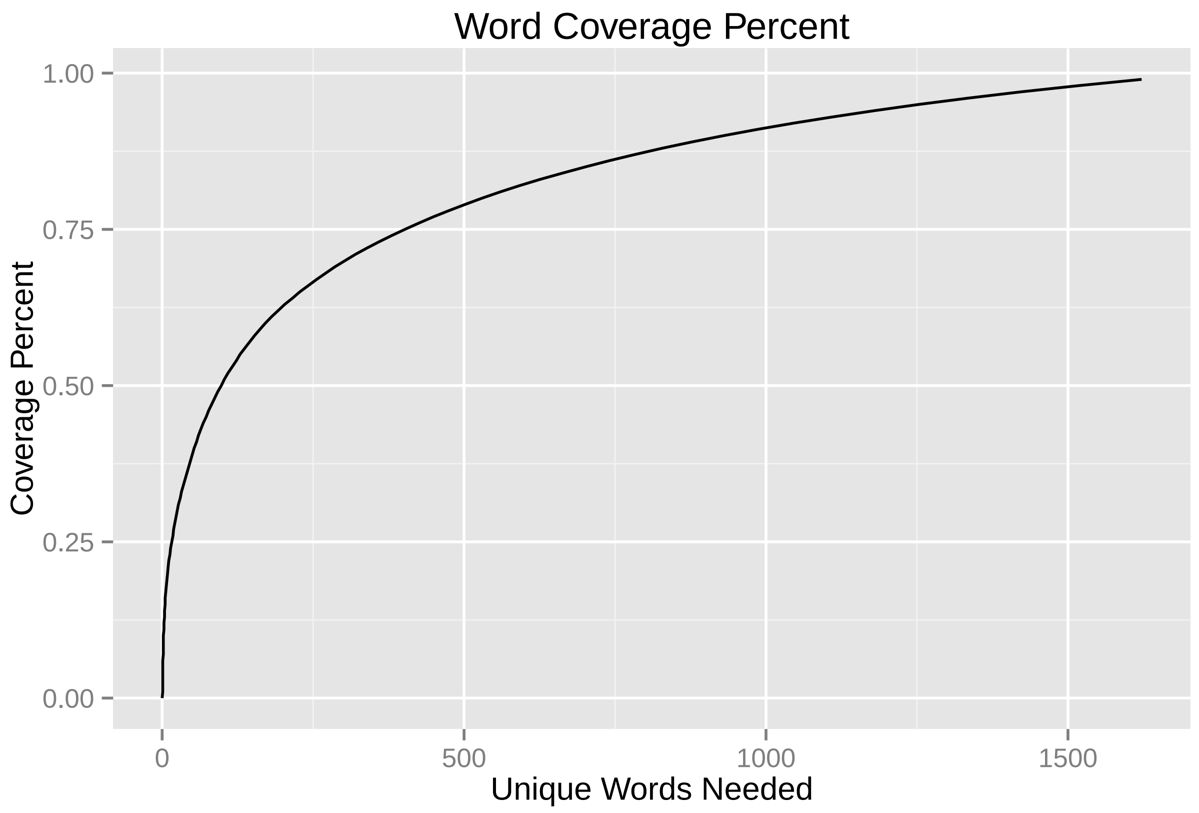
Finally, we create a function to calculate the number of unique words needed to cover x% of word instances in the sample dataset.

```
word_total <- sum(data_uni_rank$freq)

word_num_f <- function(covered_percent) {
  cum_percent = 0
  i = 1
  while (cum_percent < covered_percent) {
    cum_percent = cum_percent + data_uni_rank$freq[i]/word_total
    i = i + 1
    }
  return(i-1)
  }
```

We iterate through the number of unique words required for x% coverage. We find around 98 words are needed to achieve 50% coverage and around 930 words are needed to acheive 90% coverage in the sample dataset.

```
library("ggplot2")
seq_percent <- seq(0, 0.99, by = 0.01)
word_num <- unlist(lapply(seq_percent, word_num_f))
# plot the sequence
qplot(word_num, seq_percent, geom="line", xlab="Unique Words Needed", ylab="Coverage Percent", main="Wo
```



Findings and Conclusions

- We find some of the most frequent words are stop words which do not have much meaning, but and we choose to leave them in corpus since they are still meaningful in terms of perdicting users' input pattern.
- We also find that R has some performace issues with dealing with "big"" data. That's why for the exploratory analysis we only pick up 1% of data from the blogs, news and twitter files. The corpus is very large and we have to remove sparse terms in the document term matrix to avoid memory overflow when generating the N-Grams matrix. The performace issue will be critical we when deploy the application on Shiny App. On the other hand, small samples of data may introduce bias so there should be a balance between accuracy and performance.
- We also think about some additional improvements. For example, removing words other than English. One way is to use a complete English dictionary and remove any words that do not appear in it. We decide to keep foreign words in the corpus since most of the time they have a meaningful impact in the context.
- Another potential improvement is to increase the word coverage by introducing some external dictionary with high frequency alternatives. However this might be out of scope for the exploratory analysis phase and will be considered in the model tuning phase.

Future Steps

After the exploratory analysis phase and we have the following action items in the modeling and post-processing phase.

1. Split the large dataset into training / testing / validation.
2. Explore the number of parameters in the model and train the model to predict the most common words associated with previous input words.
3. Choose the optimal number of words in the N-Grams model to balance the accuacury "vs" performance.
4. Smooth the probabilities when a certain N-Grams isn't observed in the data.
5. Use backoff models to estimate the probability of unobserved N-Grams.
6. Test the accuracy and performace of the model based on testing dataset.
7. Design the Shiny App using the built model.
8. Prepare the slide deck for the model.