Synopsis

This is the Milestone Report for the Coursera Data Science Capstone project. The goal of the capstone project is to create a predictive text model using a large text corpus of documents as training data. Natural language processing techniques will be used to perform the analysis and build the predictive model.

This milestone report describes the major features of the training data with our exploratory data analysis and summarizes our plans for creating the predictive model.

Getting the data

```
## Load CRAN modules
library(downloader)
library(plyr);
library(dplyr)
##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:plyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
library(knitr)
## Warning: package 'knitr' was built under R version 4.0.3
library(tm)
## Warning: package 'tm' was built under R version 4.0.3
## Loading required package: NLP
## Warning: package 'NLP' was built under R version 4.0.3
## Step 1: Download the dataset and unzip folder
## Check if directory already exists?
if(!file.exists("./projectData")){
  dir.create("./projectData")
}
Url <- "https://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-
SwiftKey.zip"
## Check if zip has already been downloaded in projectData directory?
if(!file.exists("./projectData/Coursera-SwiftKey.zip")){
  download.file(Url,destfile="./projectData/Coursera-SwiftKey.zip",mode =
"wb")
}
## Check if zip has already been unzipped?
if(!file.exists("./projectData/final")){
  unzip(zipfile="./projectData/Coursera-SwiftKey.zip",exdir="./projectData")
}
```

Once the dataset is downloaded start reading it as this a huge dataset so we'll read line by line only the amount of data needed before doing that lets first list all the files in the directory List all the files of /final/en_US Dataset folder The data sets consist of text from 3 different sources: 1) News, 2) Blogs and 3) Twitter feeds. In this project, we will only focus on the English - US data sets.

```
path <- file.path("./projectData/final" , "en_US")
files<-list.files(path, recursive=TRUE)
# Lets make a file connection of the twitter data set
con <- file("./projectData/final/en_US/en_US.twitter.txt", "r")
#lineTwitter<-readLines(con,encoding = "UTF-8", skipNul = TRUE)
lineTwitter<-readLines(con, skipNul = TRUE)
# Close the connection handle when you are done
close(con)
# Lets make a file connection of the blog data set
con <- file("./projectData/final/en_US/en_US.blogs.txt", "r")
#lineBlogs<-readLines(con,encoding = "UTF-8", skipNul = TRUE)
lineBlogs<-readLines(con, skipNul = TRUE)
# Close the connection handle when you are done
close(con)
# Lets make a file connection of the news data set
con <- file("./projectData/final/en_US/en_US.news.txt", "r")
#lineNews<-readLines(con,encoding = "UTF-8", skipNul = TRUE)
lineNews<-readLines(con, skipNul = TRUE)
## Warning in readLines(con, skipNul = TRUE): incomplete final line found on
'./
## projectData/final/en_US/en_US.news.txt'
# Close the connection handle when you are done
close(con)
```

We examined the data sets and summarize our findings (file sizes, line counts, word counts, and mean words per line) below.

```
library(stringi)
# Get file sizes
lineBlogs.size <- file.info("./projectData/final/en_US/en_US.blogs.txt")$size
/ 1024 ^ 2
lineNews.size <- file.info("./projectData/final/en_US/en_US.news.txt")$size /
1024 ^ 2
lineTwitter.size <-
file.info("./projectData/final/en_US/en_US.twitter.txt")$size / 1024 ^ 2
# Get words in files
lineBlogs.words <- stri_count_words(lineBlogs)
lineNews.words <- stri_count_words(lineNews)
lineTwitter.words <- stri_count_words(lineTwitter)
# Summary of the data sets
data.frame(source = c("blogs", "news", "twitter"),
           file.size.MB = c(lineBlogs.size, lineNews.size, lineTwitter.size),
           num.lines = c(length(lineBlogs), length(lineNews),
length(lineTwitter)),
           num.words = c(sum(lineBlogs.words), sum(lineNews.words),
sum(lineTwitter.words)),
           mean.num.words = c(mean(lineBlogs.words), mean(lineNews.words),
mean(lineTwitter.words)))
```

```
##    source file.size.MB num.lines num.words mean.num.words
## 1   blogs     200.4242    899288  38154238       42.42716
## 2    news     196.2775     77259   2693898       34.86840
## 3 twitter     159.3641   2360148  30218166       12.80350
```

Cleaning The Data

Before performing exploratory analysis, we must clean the data first. This involves removing URLs, special characters, punctuations, numbers, excess whitespace, stopwords, and changing the text to lower case. Since the data sets are quite large, we will randomly choose 2% of the data to demonstrate the data cleaning and exploratory analysis also please take care of the UTF chars.

```
library(tm)
# Sample the data
set.seed(5000)
data.sample <- c(sample(lineBlogs, length(lineBlogs) * 0.02),
                 sample(lineNews, length(lineNews) * 0.02),
                 sample(lineTwitter, length(lineTwitter) * 0.02))
# Create corpus and clean the data
corpus <- VCorpus(VectorSource(data.sample))
toSpace <- content_transformer(function(x, pattern) gsub(pattern, " ", x))
corpus <- tm_map(corpus, toSpace, "(f|ht)tp(s?)://(.*)[.][a-z]+")
corpus <- tm_map(corpus, toSpace, "@[^\\s]+")
corpus <- tm_map(corpus, tolower)
corpus <- tm_map(corpus, removeWords, stopwords("en"))
corpus <- tm_map(corpus, removePunctuation)
corpus <- tm_map(corpus, removeNumbers)
corpus <- tm_map(corpus, stripWhitespace)
corpus <- tm_map(corpus, PlainTextDocument)
```

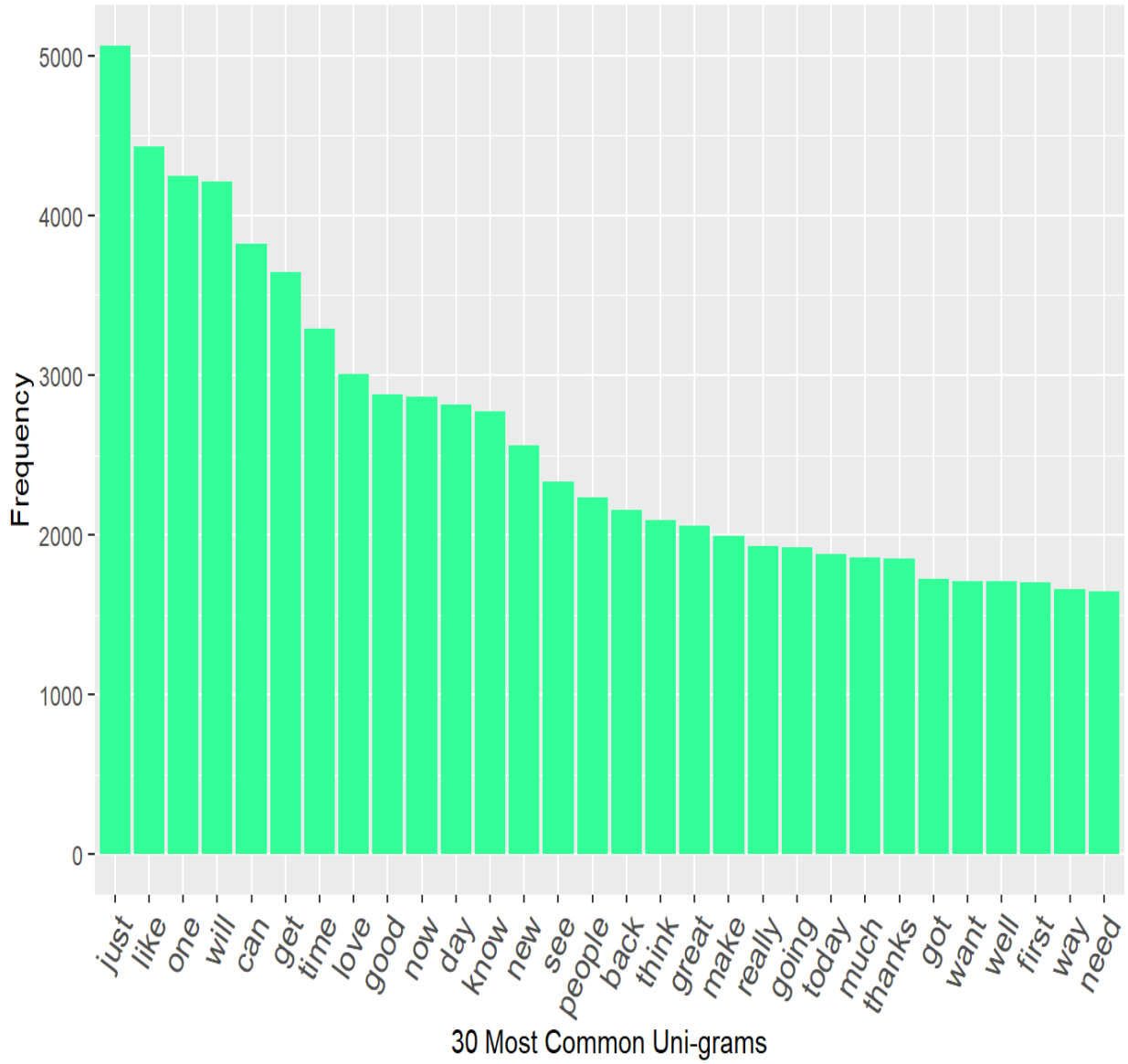
##Exploratory Analysis Now tine to do some exploratory analysis on the data. It would be interesting and helpful to find the most frequently occurring words in the data. Here we list the most common (n-grams) uni-grams, bi-grams, and tri-grams.

```
library(RWeka)
## Warning: package 'RWeka' was built under R version 4.0.3
## java.home option:
## JAVA_HOME environment variable: C:\Program Files\Java\jre7
## Warning in fun(libname, pkgname): Java home setting is INVALID, it will be
ignored.
## Please do NOT set it unless you want to override system settings.
library(ggplot2)
##
## Attaching package: 'ggplot2'
## The following object is masked from 'package:NLP':
##
##     annotate
##annotate
options(mc.cores=1)
# we'll get the frequencies of the word
getFreq <- function(tdm) {
  freq <- sort(rowSums(as.matrix(tdm)), decreasing = TRUE)
  return(data.frame(word = names(freq), freq = freq))
```

```
}
bigram <- function(x) NGramTokenizer(x, Weka_control(min = 2, max = 2))
trigram <- function(x) NGramTokenizer(x, Weka_control(min = 3, max = 3))
makePlot <- function(data, label) {
  ggplot(data[1:30,], aes(reorder(word, -freq), freq)) +
         labs(x = label, y = "Frequency") +
         theme(axis.text.x = element_text(angle = 60, size = 12, hjust = 1))
+
         geom_bar(stat = "identity", fill = I("#33ff99"))
}
# Get frequencies of most common n-grams in data sample
freq1 <- getFreq(removeSparseTerms(TermDocumentMatrix(corpus), 0.9999))
freq2 <- getFreq(removeSparseTerms(TermDocumentMatrix(corpus, control =
list(tokenize = bigram)), 0.9999))
freq3 <- getFreq(removeSparseTerms(TermDocumentMatrix(corpus, control =
list(tokenize = trigram)), 0.9999))
```

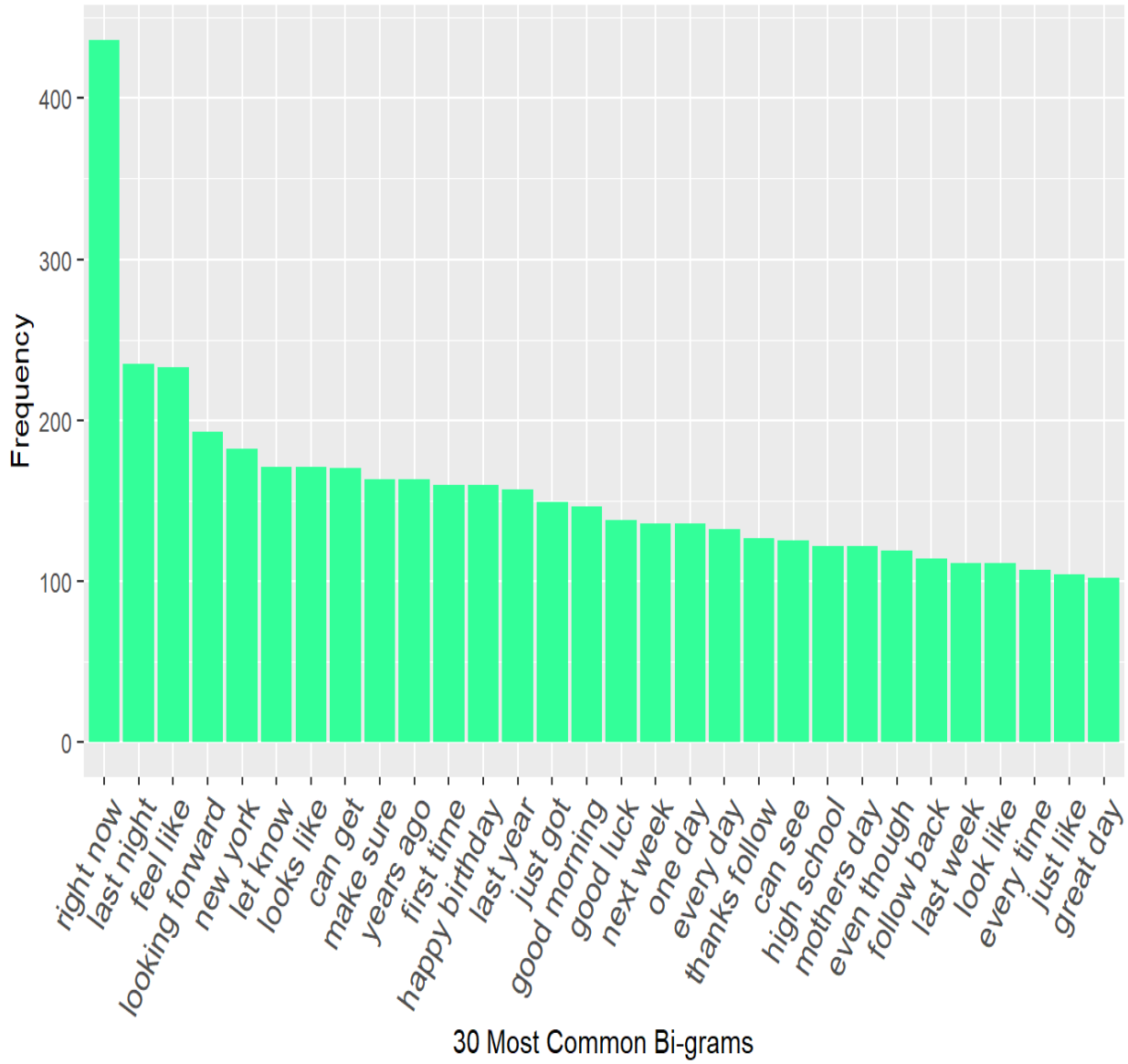
Here is a histogram of the 30 most common unigrams in the data sample.

```
makePlot(freq1, "30 Most Common Uni-grams")
```



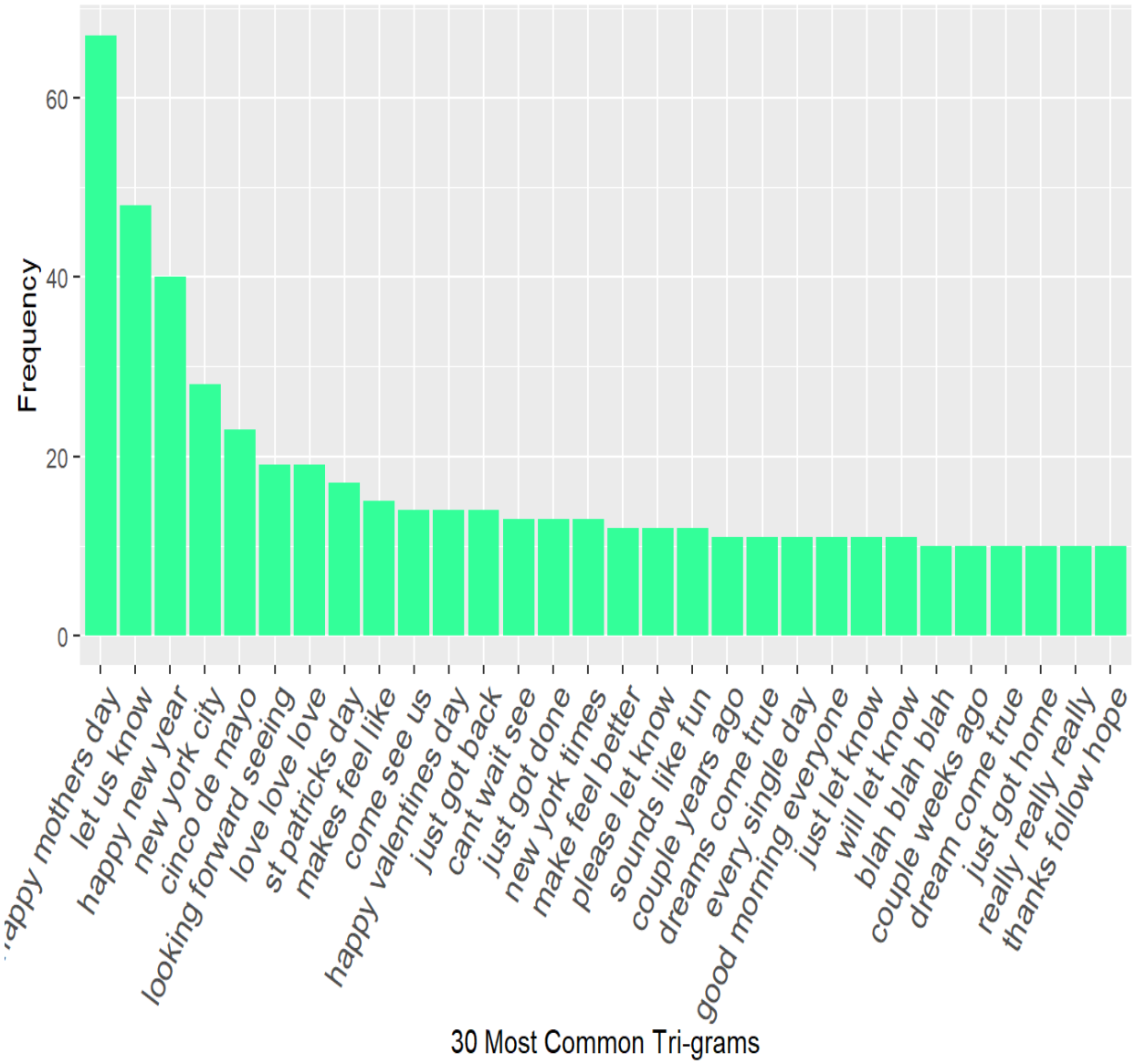
Here is a histogram of the 30 most common bigrams in the data sample.

```
makePlot(freq2, "30 Most Common Bi-grams")
```



Here is a histogram of the 30 most common trigrams in the data sample.

```
makePlot(freq3, "30 Most Common Tri-grams")
```



Conclusion and further planning

This concludes our exploratory analysis. The next steps of this capstone project would be to finalize our predictive algorithm, and deploy our algorithm as a Shiny app.

Our predictive algorithm will be using n-gram model with frequency lookup similar to our exploratory analysis above. One possible strategy would be to use the trigram model to predict the next word. If no matching trigram can be found, then the algorithm would back off to the bigram model, and then to the unigram model if needed.

The user interface of the Shiny app will consist of a text input box that will allow a user to enter a phrase. Then the app will use our algorithm to suggest the most likely next word after a short delay.