Milestone Report

Overview

This document contains the Milestone Report for the the Data Science Specialization on Coursera. Sections are organized according to the tasks in the course - specifically, Task 1: Getting and Cleaning the Data, and Task 2: Exploratory Data Analysis.

Start with some setup code. Message set to F, to silence package messages.

```
rm(list=ls())
set.seed(1729)
library(dplyr)
library(RWeka)
```

Task 1: Getting and Cleaning the Data

Download the zip file from the link provided in the course if it is not already in the local directory. Similarly, if the unzipped data is not in the local directory, unzip the data file.

```
if (!file.exists('data.zip')) {
  download.file('https://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-S
wiftKey.zip', 'data.zip')
  }
if (!dir.exists('final')) {
  unzip('data.zip')
  }
```

Read the data into R. Warning set to F, to silence NA warnings.

```
bRaw <- readLines('final/en_US/en_US.blogs.txt')
nRaw <- readLines('final/en_US/en_US.news.txt')
tRaw <- readLines('final/en_US/en_US.twitter.txt')
```

Do the quiz questions, as they largely concern the entire data set.

```
length(tRaw)
```

```
## [1] 2360148
```

```
max(sapply(tRaw, nchar), sapply(bRaw, nchar), sapply(nRaw, nchar))
```

```
## [1] 40835
```

```
sum(grepl('love', tRaw))/sum(grepl('hate', tRaw))
```

```
## [1] 4.108592
```

```
tRaw[grepl('biostats', tRaw)]
```

```
## [1] "i know how you feel.. i have biostats on tuesday and i have yet to study =/"
```

```
sum(grepl('A computer once beat me at chess, but it was no match for me at kickboxing
', tRaw))
```

```
## [1] 3
```

Subset the data to a more workable size.

```
bRaw <- bRaw[rbinom(length(bRaw), 1, 0.01)==1]
nRaw <- nRaw[rbinom(length(nRaw), 1, 0.01)==1]
tRaw <- tRaw[rbinom(length(tRaw), 1, 0.01)==1]
```

Clean the data, converting to lowercase and removing all characters besides letters and numbers.

```
cleanFunc <- function(x) as.character(gsub("[^a-z0-9 ]*", "", tolower(x)))
b <- sapply(bRaw, cleanFunc, USE.NAMES = F)
n <- sapply(nRaw, cleanFunc, USE.NAMES = F)
t <- sapply(tRaw, cleanFunc, USE.NAMES = F)
```

Separate the data into training and test sets (80/20), and combine the training data from all sources into a unified training set. Keep test sets separate, to assess accuracy on a per-source basis later.

```
bMask <- rbinom(length(b), 1, 0.8)
nMask <- rbinom(length(n), 1, 0.8)
tMask <- rbinom(length(t), 1, 0.8)
train <- c(b[bMask==1], n[nMask==1], t[tMask==1])
bTest <- b[bMask==0]
nTest <- n[nMask==0]
tTest <- t[tMask==0]
```

Tokenize the data into 1-grams (words), 2-grams (pairs), 3-grams (trips), and 4-grams(quads)

```
words <- unlist(sapply(train, WordTokenizer, USE.NAMES = F))
pairs <- unlist(sapply(train, function(x) NGramTokenizer(x, Weka_control(min=2, max=
2)), USE.NAMES = F))
trips <- unlist(sapply(train, function(x) NGramTokenizer(x, Weka_control(min=3, max=
3)), USE.NAMES = F))
quads <- unlist(sapply(train, function(x) NGramTokenizer(x, Weka_control(min=4, max=
4)), USE.NAMES = F))
```

Task 2: Exploratory Data Analysis

Count of the n-grams in the training data.

```
length(words)
```

```
## [1] 557088
```

```
length(pairs)
```

```
## [1] 530176
```

```
length(trips)
```

```
## [1] 503369
```

```
length(quads)
```

```
## [1] 477369
```

Measure the average lengths of each type of n-gram.

```
mean(sapply(words, nchar))
```

```
## [1] 4.352499
```

```
mean(sapply(pairs, nchar))
```

```
## [1] 9.668369
```

```
mean(sapply(trips, nchar))
```

```
## [1] 14.99502
```

```
mean(sapply(quads, nchar))
```

```
## [1] 20.33863
```

Convert each list of n-grams to a data frame, using gsub to isolate and record each of the component words.

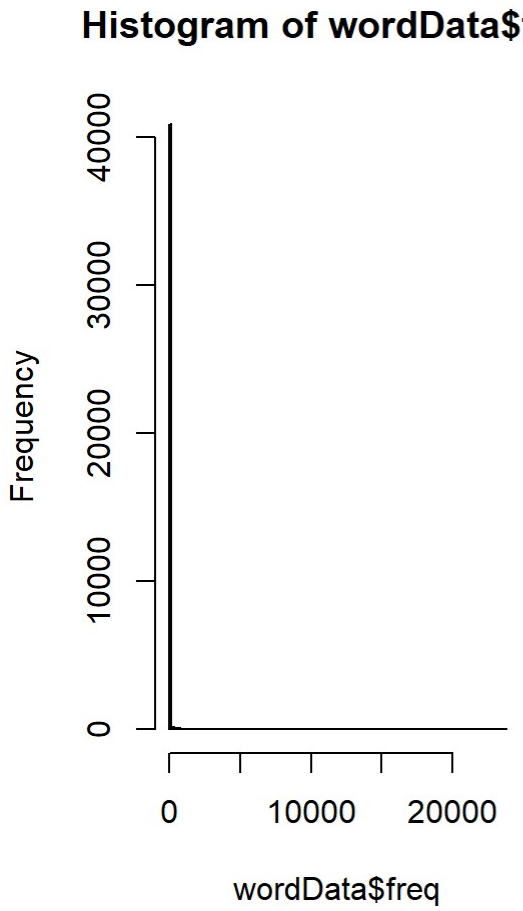
```
str1Func <- function(x) gsub(' [a-z1-9]*?$', '', x)
str2Func <- function(x) gsub('^.* ', '', x)
wordData <- data.frame(str=words)
pairData <- data.frame(str=pairs)
pairData$str1 <- sapply(pairData$str, str1Func, USE.NAMES = F)
pairData$str2 <- sapply(pairData$str, str2Func, USE.NAMES = F)
tripData <- data.frame(str=trips)
tripData$str1 <- sapply(tripData$str, str1Func, USE.NAMES = F)
tripData$str2 <- sapply(tripData$str, str2Func, USE.NAMES = F)
quadData <- data.frame(str=quads)
quadData$str1 <- sapply(quadData$str, str1Func, USE.NAMES = F)
quadData$str2 <- sapply(quadData$str, str2Func, USE.NAMES = F)
```

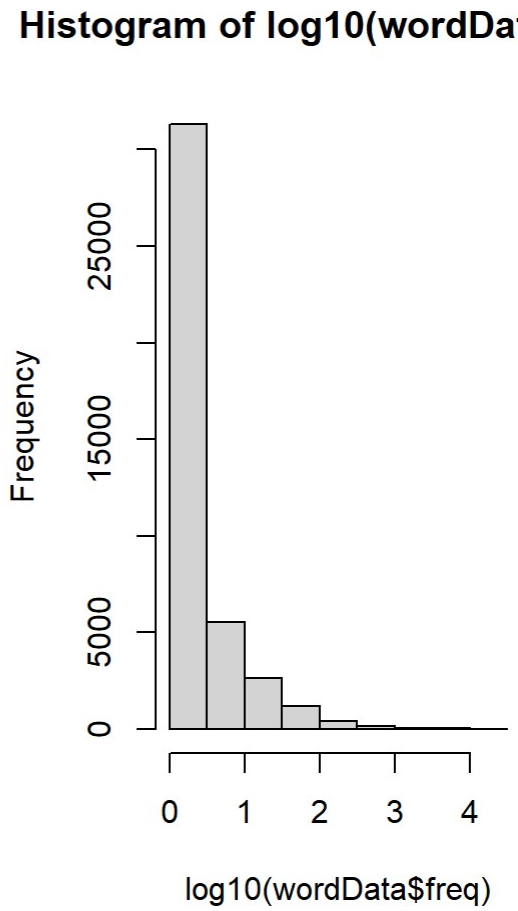
Calculate the frequency of each n-gram by collapsing identical rows into groups and counting the number of instances in each group. Message = F to silence dplyr regrouping message.

```
wordData <- arrange(summarise(group_by(wordData, str), freq=n()), desc(freq))
pairData <- arrange(summarise(group_by(pairData, str, str1, str2), freq=n()), desc(fr
eq))
tripData <- arrange(summarise(group_by(tripData, str, str1, str2), freq=n()), desc(fr
eq))
quadData <- arrange(summarise(group_by(quadData, str, str1, str2), freq=n()), desc(fr
eq))
```

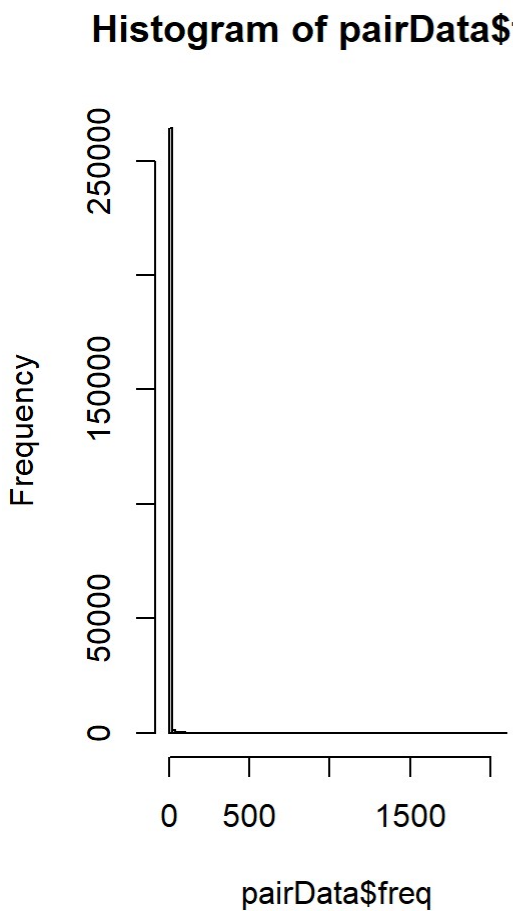
Plot the resulting frequency distributions, on ordinary and semi-log histograms.

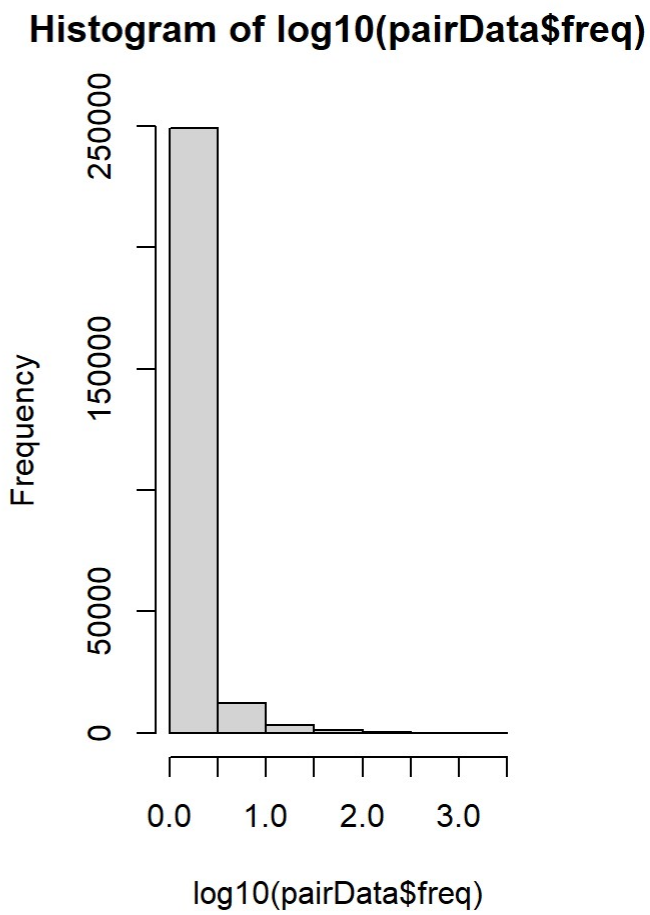
```
par(mar=c(4,4,4,4), mfrow=c(1,2))
hist(wordData$freq, breaks=100)
hist(log10(wordData$freq), breaks=10)
```



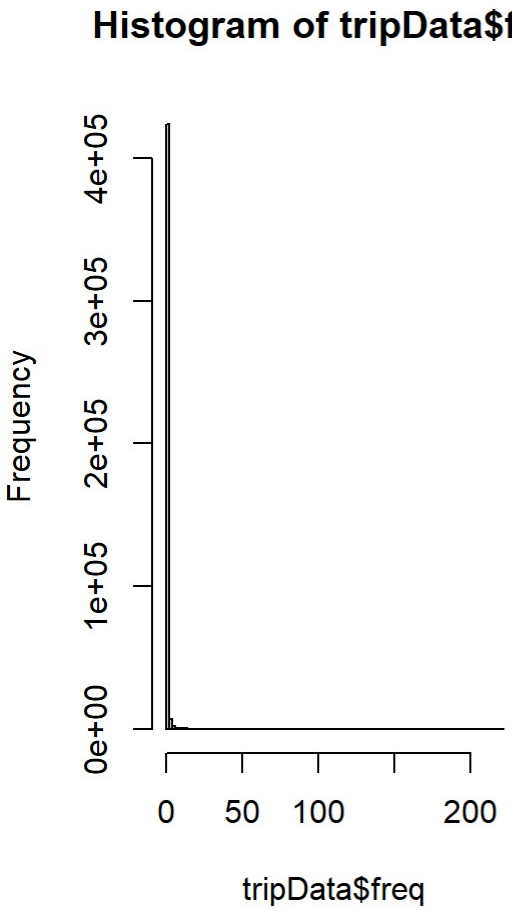


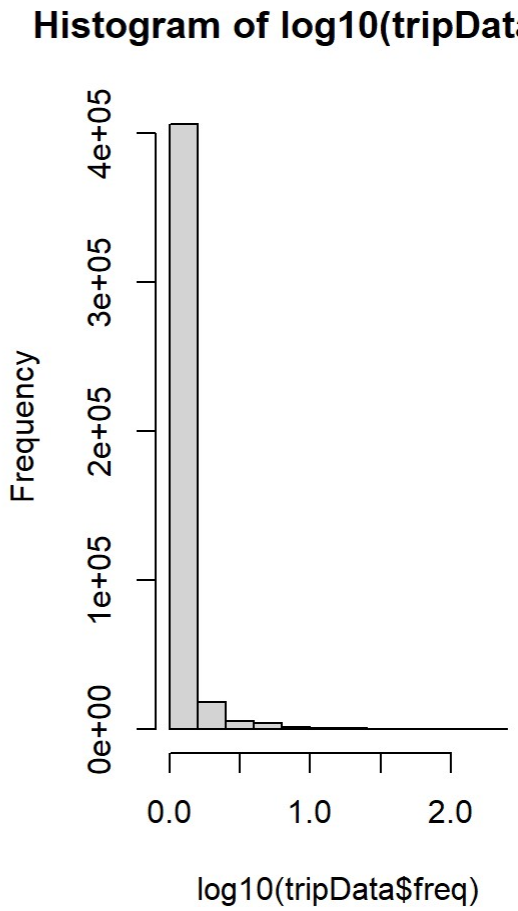

```
hist(pairData$freq, breaks=100)
hist(log10(pairData$freq), breaks=10)
```



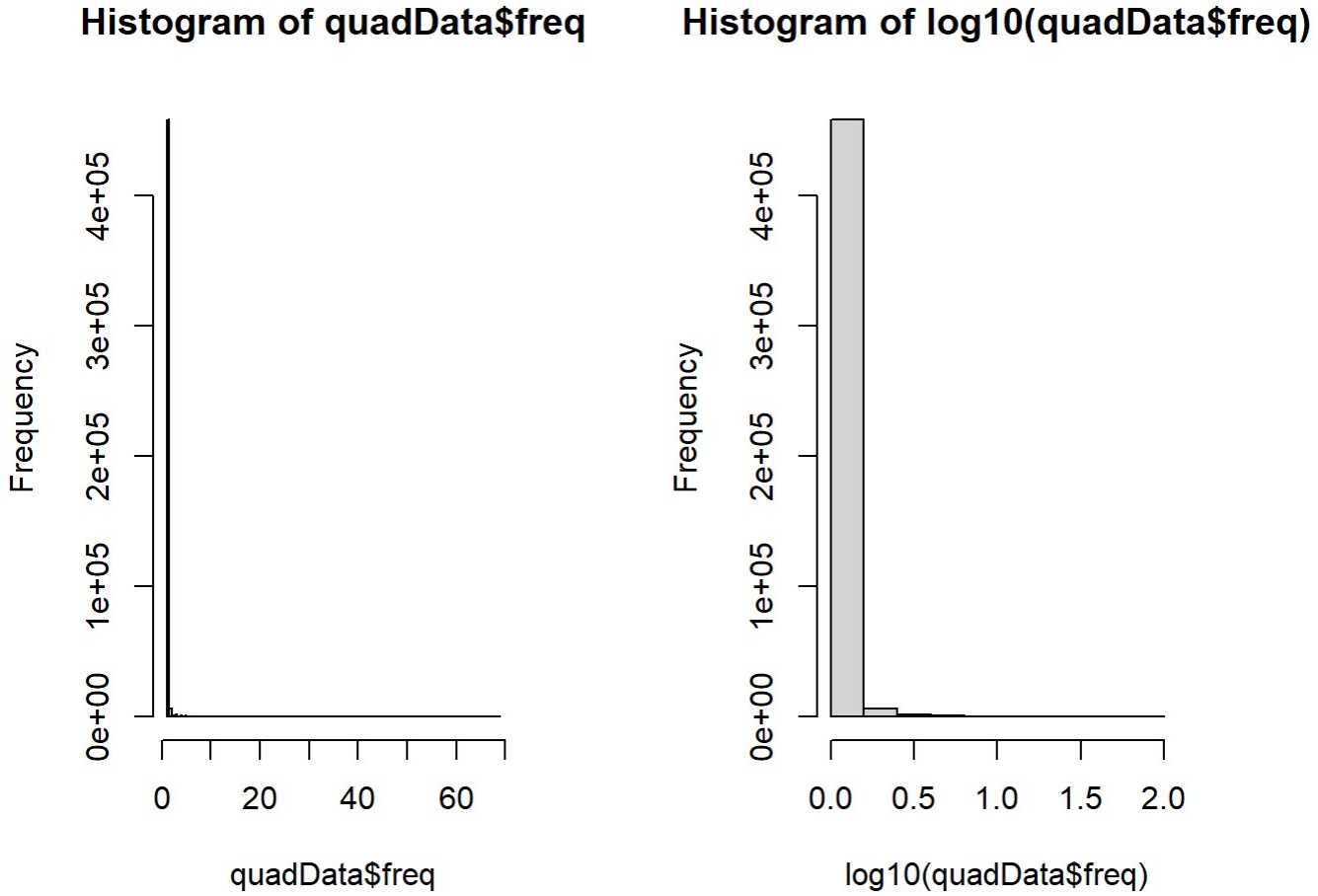



```
hist(tripData$freq, breaks=100)
hist(log10(tripData$freq), breaks=10)
```



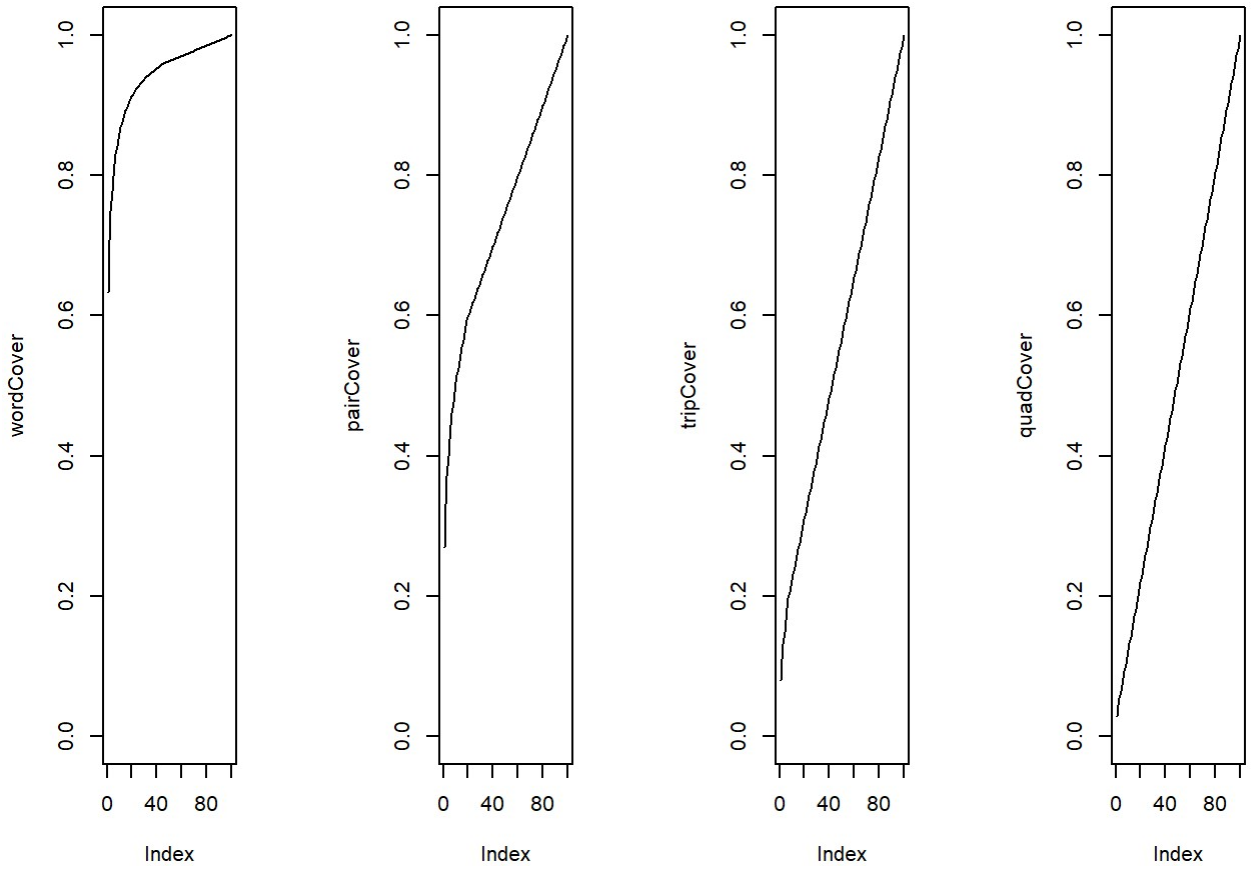



```
hist(quadData$freq, breaks=100)
hist(log10(quadData$freq), breaks=10)
```



Plot the coverage of all the instances of words in the data set as a function of the percent of unique words included, starting with the most common.

```
cover <- function(x) {
  tot <- sum(x$freq)
  len <- length(x$freq)
  sapply(1:100, function(p) sum(x[1:round(len*p/100), 'freq'])/tot)
}
par(mar=c(4,4,4,4), mfrow=c(1,4))
wordCover <- cover(wordData)
pairCover <- cover(pairData)
tripCover <- cover(tripData)
quadCover <- cover(quadData)
plot(wordCover, type='l', ylim=c(0,1))
plot(pairCover, type='l', ylim=c(0,1))
plot(tripCover, type='l', ylim=c(0,1))
plot(quadCover, type='l', ylim=c(0,1))
```



Plans for Modeling

To try and keep the model fast enough to train and reference to be practical, I plan to calculate only the most likely next word for each n-gram, not an entire matrix of conditional probabilities. This will reduce the size of the model from scaling quadratically to linearly with volume of training data, while discarding little information that will actually be used for prediction. Hopefully, this will allow more data to be used, and lead to a better model.