Data Science Capstone

Overview

This document contains Tasks 1-5 of the Capstone Project, as part of the Data Science Specialization on Coursera. Sections are organized according to the tasks in the course. Three tasks are not included - Task 0: Understanding the Problem, which doesn't involve any code, Task 6: Data Product, which must be built in a separate Shiny app for speed, and Task 7: Slide Deck, which must be a separate document.

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

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

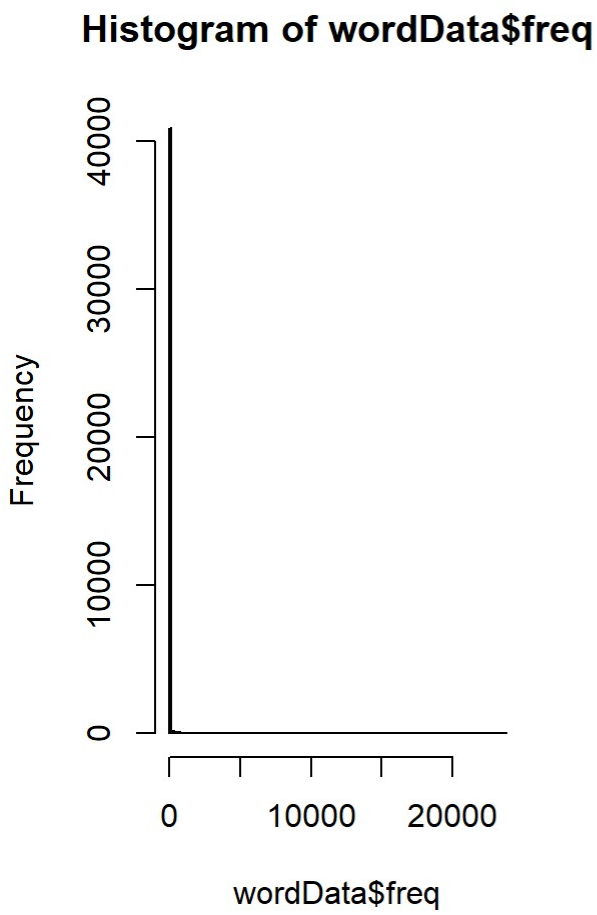
```
str1Func <- function(x) gsub(' [a-z1-9]*?$', '', x)
str2Func <- function(x) gsub('^.* ', '', x)
wordData <- data.frame(str=words)
pairData <- data.frame(str=pairs)
pairData$str1 <- sapply(pairData$str, str1Func, USE.NAMES = F)
pairData$str2 <- sapply(pairData$str, str2Func, USE.NAMES = F)
tripData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

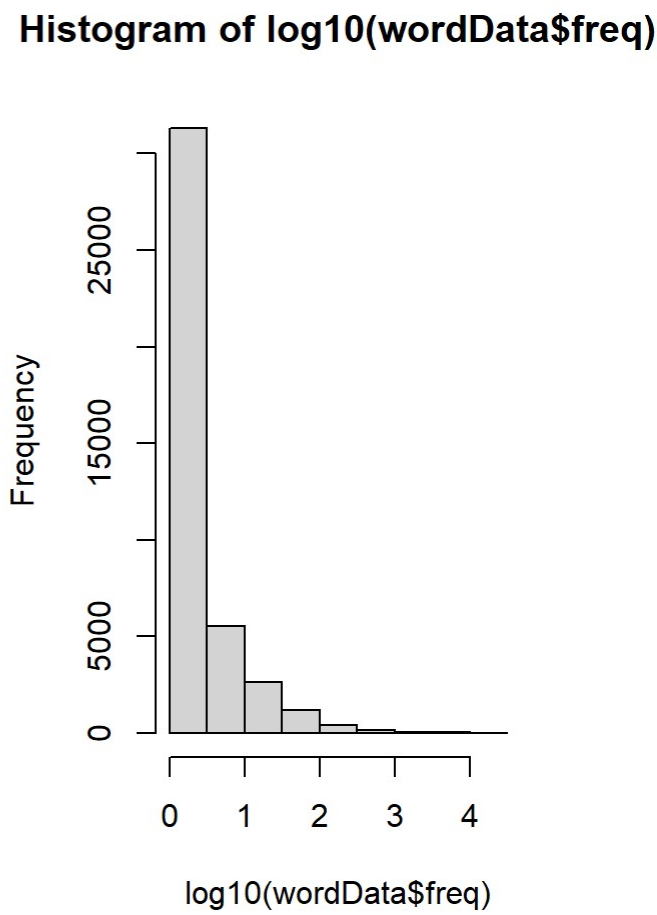
Calculate the frequency of each n-gram by collapsing identical rows into groups and counting the number of instances in each group. Message = F to silence dplyr regrouping message.

```
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))
```

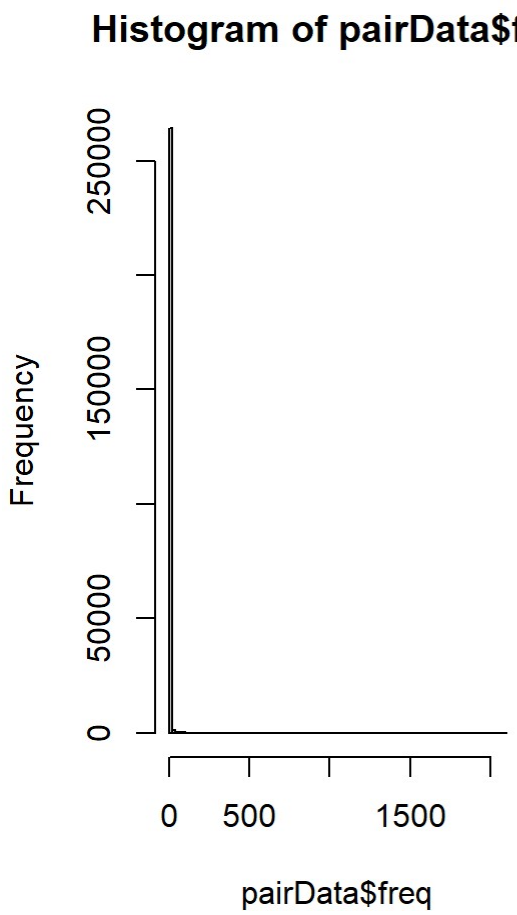
Plot the resulting frequency distributions, on ordinary and semi-log histograms.

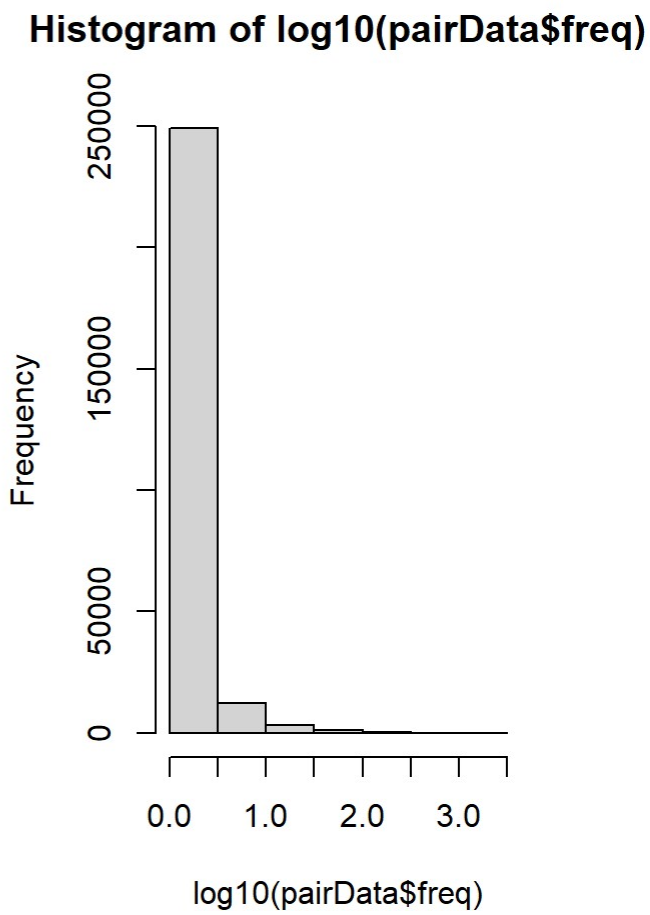
```
par(mar=c(4,4,4,4), mfrow=c(1,2))
hist(wordData$freq, breaks=100)
hist(log10(wordData$freq), breaks=10)
```



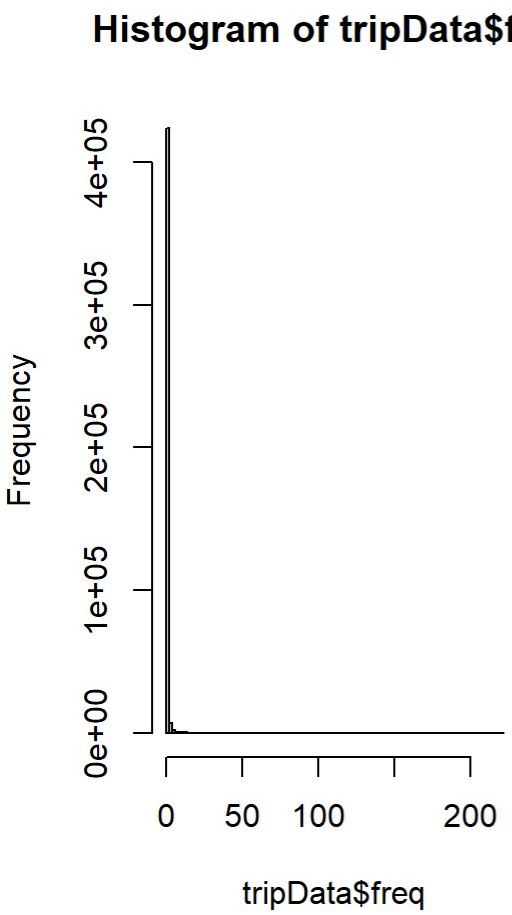


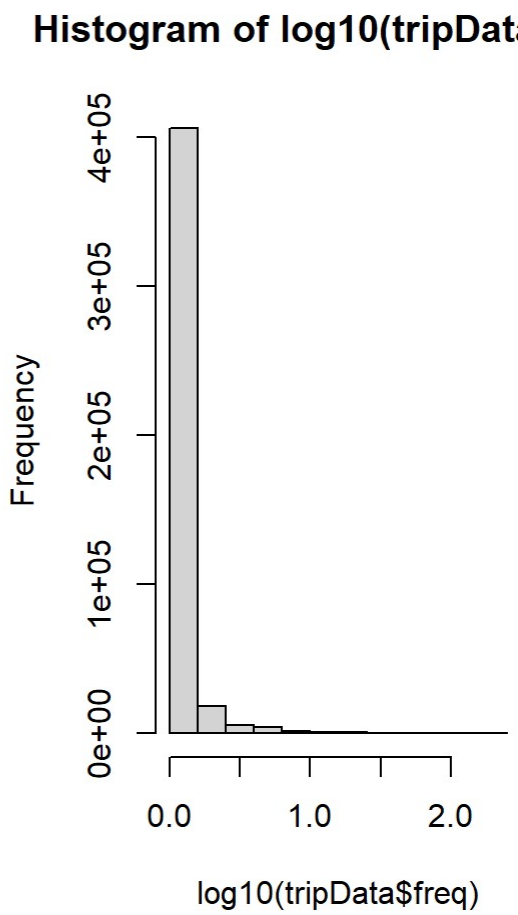

```
hist(pairData$freq, breaks=100)
hist(log10(pairData$freq), breaks=10)
```



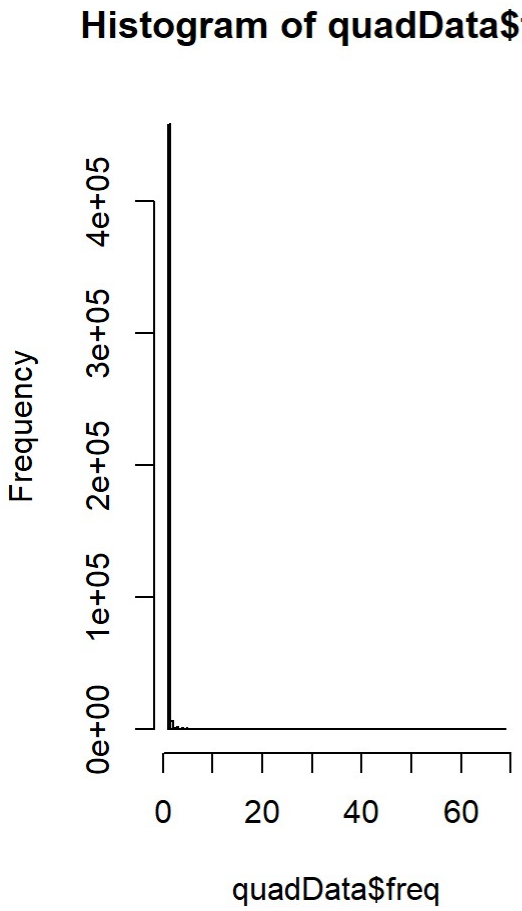


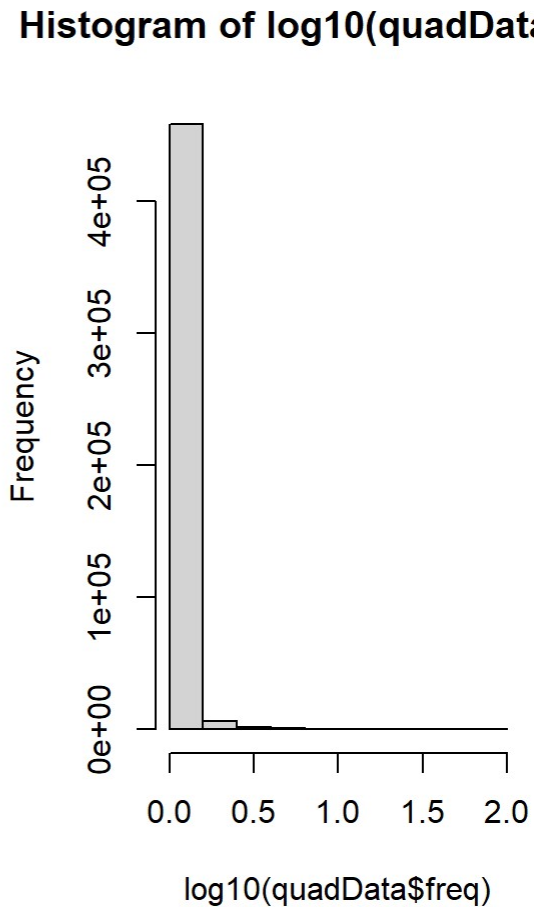

```
hist(tripData$freq, breaks=100)
hist(log10(tripData$freq), breaks=10)
```





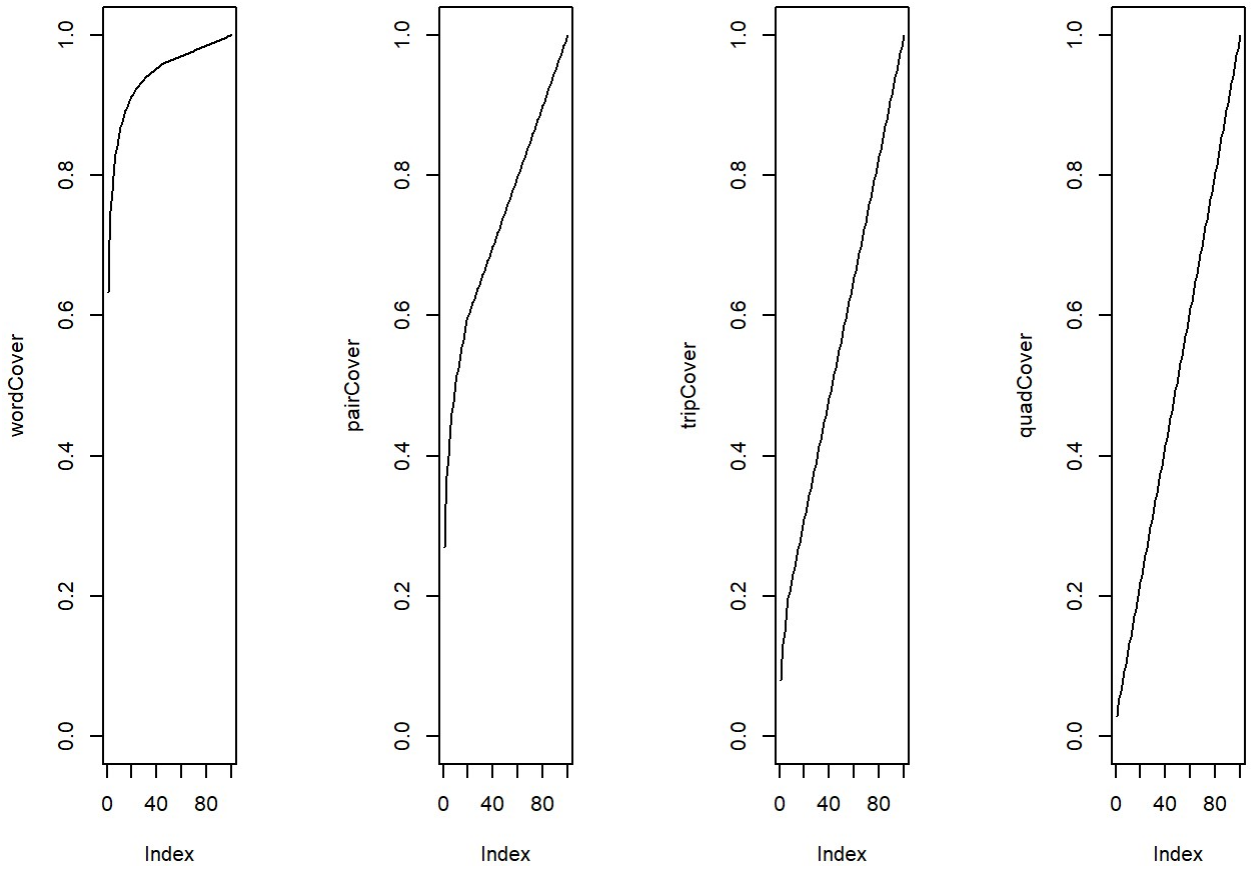

```
hist(quadData$freq, breaks=100)
hist(log10(quadData$freq), breaks=10)
```





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))
```



Task 3: Modeling

Build a simple model by simply storing the most common next word for a given n-gram as an additional field in the n-gram data frame. Since the goal is to determine the most likely next word, and not necessarily to calculate the conditional probabilities for each word following each n-gram, this is a much faster way to produce predictions.

```
wordData$nextWord <- sapply(wordData$str, function(x) as.character(pairData[pairDat
a$str1==x, 'str2'][1, ]))
pairData$nextWord <- sapply(pairData$str, function(x) as.character(tripData[tripDat
a$str1==x, 'str2'][1, ]))
tripData$nextWord <- sapply(tripData$str, function(x) as.character(quadData[quadDat
a$str1==x, 'str2'][1, ]))
```

Save the model, so that it can be loaded up quickly later without running the entire model-building process again.

```
saveRDS(wordData, "app/wordData.Rda")
saveRDS(pairData, "app/pairData.Rda")
saveRDS(tripData, "app/tripData.Rda")
saveRDS(quadData, "app/quadData.Rda")
```

Task 4: Prediction Model

Implement a simple back-off method to combine the three models into a single prediction model - attempt to predict with the longest n-gram available, and “back off” to shorter n-grams if there is no prediction available.

```
backoff <- function(x) {
  clean <- cleanFunc(x)
  trips <- NGramTokenizer(clean, Weka_control(min=3, max=3))
  if (length(trips) > 0) {
    lastTrip <- trips[length(trips)]
    if (lastTrip %in% tripData$str) return(as.character(tripData[tripData$str==lastTr
ip, 'nextWord'][1, ]))
  }
  pairs <- NGramTokenizer(clean, Weka_control(min=2, max=2))
  if (length(pairs) > 0) {
    lastPair <- pairs[length(pairs)]
    if (lastPair %in% pairData$str) return(as.character(pairData[pairData$str==lastPa
ir, 'nextWord'][1, ]))

  }
  words <- WordTokenizer(clean)
  if (length(words) > 0) {
    lastWord <- words[length(words)]
    if (lastWord %in% wordData$str) return(as.character(wordData[wordData$str==lastWo
rd, 'nextWord'][1, ]))
  }
  return(NA)
}
```

Prepare the test data by tokenizing it into 2,3, and 4-grams, to assess the accuracy of predictions made on 1,2, and 3-grams.

```
bPairsTest <- unlist(sapply(bTest, function(x) unlist(NGramTokenizer(x, Weka_control
(min=2, max=2))), USE.NAMES = F))
bTripsTest <- unlist(sapply(bTest, function(x) unlist(NGramTokenizer(x, Weka_control
(min=3, max=3))), USE.NAMES = F))
bQuadsTest <- unlist(sapply(bTest, function(x) unlist(NGramTokenizer(x, Weka_control
(min=4, max=4))), USE.NAMES = F))
nPairsTest <- unlist(sapply(nTest, function(x) unlist(NGramTokenizer(x, Weka_control
(min=2, max=2))), USE.NAMES = F))
nTripsTest <- unlist(sapply(nTest, function(x) unlist(NGramTokenizer(x, Weka_control
(min=3, max=3))), USE.NAMES = F))
nQuadsTest <- unlist(sapply(nTest, function(x) unlist(NGramTokenizer(x, Weka_control
(min=4, max=4))), USE.NAMES = F))
tPairsTest <- unlist(sapply(tTest, function(x) unlist(NGramTokenizer(x, Weka_control
(min=2, max=2))), USE.NAMES = F))
tTripsTest <- unlist(sapply(tTest, function(x) unlist(NGramTokenizer(x, Weka_control
(min=3, max=3))), USE.NAMES = F))
tQuadsTest <- unlist(sapply(tTest, function(x) unlist(NGramTokenizer(x, Weka_control
(min=4, max=4))), USE.NAMES = F))
```

Use the test data to determine the accuracy of the model's predictions, testing separately for each of the three data sources, as well as for predictions based on 1-grams, 2-grams, and 3-grams.

```
accTest <- function(x) sapply(x[1:min(1000, length(x))], function(y) backoff(str1Func
(y))==str2Func(y), USE.NAMES = F)
testingData <- list(bPairsTest, bTripsTest, bQuadsTest, nPairsTest, nTripsTest, nQuad
sTest, tPairsTest, tTripsTest, tQuadsTest)
separate <- sapply(testingData, accTest)
together <- unlist(separate)
```

Summarize accuracy based on the tests performed, both overall and in matrices breaking down accuracy rates by source and by n-gram size.

```
correctRate <-function(x) sum(x==T, na.rm = T)/length(x)
naRate <- function(x) sum(is.na(x))/length(x)
correctRate(together)
```

```
## [1] 0.1031111
```

```
naRate(together)
```

```
## [1] 0.05977778
```

```
correctMatrix <- matrix(apply(separate, 2, correctRate), nrow=3)
rownames(correctMatrix) <- c(1,2,3)
colnames(correctMatrix) <- c('b', 'n', 't')
correctMatrix
```

```
##       b     n     t
## 1 0.103 0.110 0.089
## 2 0.106 0.104 0.110
## 3 0.103 0.103 0.100
```

```
naMatrix <- matrix(apply(separate, 2, naRate), nrow=3)
rownames(naMatrix) <- c(1,2,3)
colnames(naMatrix) <- c('b', 'n', 't')
naMatrix
```

```
##       b     n     t
## 1 0.049 0.069 0.048
## 2 0.059 0.070 0.061
## 3 0.061 0.061 0.060
```

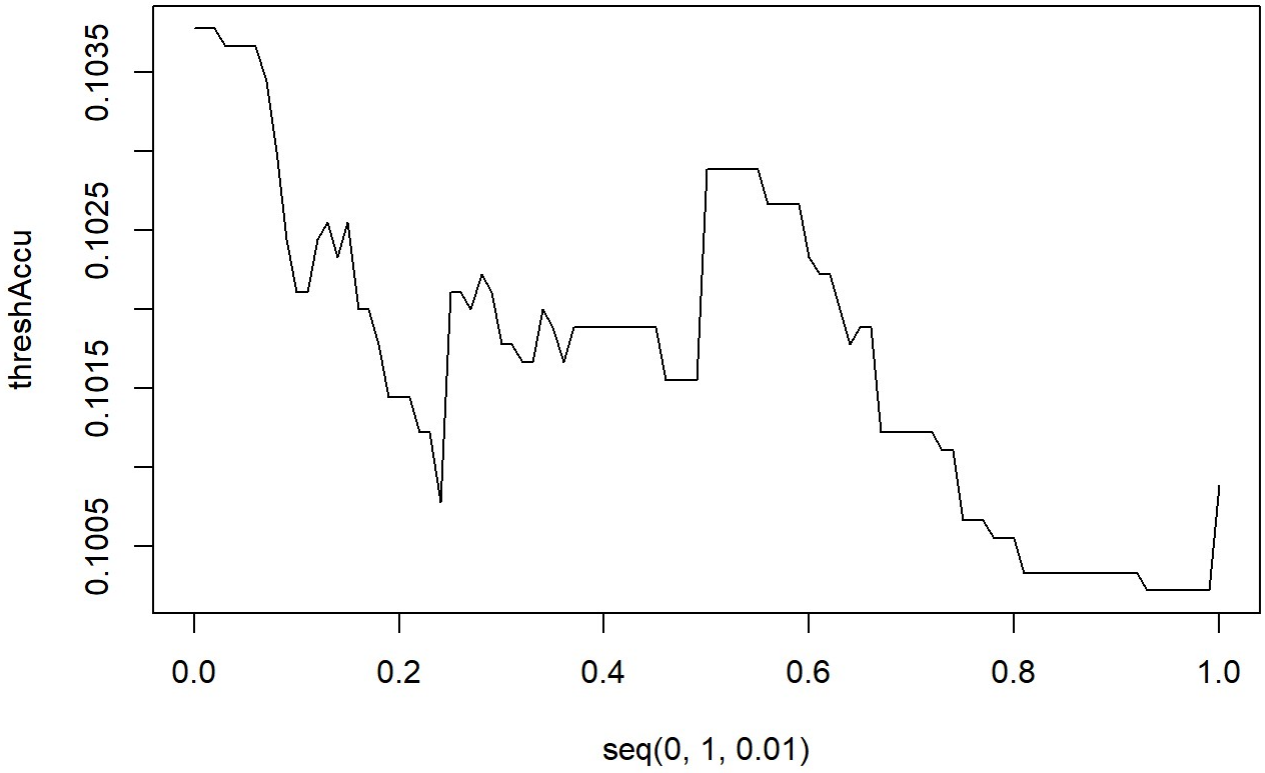
Task 5: Creative Exploration

Implement a slightly less simple back-off method that will back off not only in response to no prediction, but also in response to a weak prediction (less than n% conditional probability).

```
backoff2 <- function(x, n) {
  clean <- cleanFunc(x)
  trips <- NGramTokenizer(clean, Weka_control(min=3, max=3))
  if (length(trips) > 0) {
    lastTrip <- trips[length(trips)]
    topFreq <- quadData[quadData$str1==lastTrip, 'freq'][1, ]
    botFreq <- tripData[tripData$str==lastTrip, 'freq'][1, ]
    if (lastTrip %in% tripData$str & !is.na(topFreq/botFreq) & topFreq/botFreq > n) {
      return(as.character(tripData[tripData$str==lastTrip, 'nextWord'][1, ]))
    }
  }
  pairs <- NGramTokenizer(clean, Weka_control(min=2, max=2))
  if (length(pairs) > 0) {
    lastPair <- pairs[length(pairs)]
    topFreq <- tripData[tripData$str1==lastPair, 'freq'][1, ]
    botFreq <- pairData[pairData$str==lastPair, 'freq'][1, ]
    if (lastPair %in% pairData$str & !is.na(topFreq/botFreq) & as.numeric(topFreq/bot
Freq) > n) {
      return(as.character(pairData[pairData$str==lastPair, 'nextWord'][1, ]))
    }
  }
  words <- WordTokenizer(clean)
  if (length(words) > 0) {
    lastWord <- words[length(words)]
    if (lastWord %in% wordData$str) {
      return(as.character(wordData[wordData$str==lastWord, 'nextWord'][1, ]))
    }
  }
  return(NA)
}
```

Test new back-off method, scanning across possible values of n to see which provides best results.

```
accTest2 <- function(x, n) sapply(x[1:min(1000, length(x))], function(y) backoff2(str
1Func(y), n)==str2Func(y), USE.NAMES = F)
threshAccu <- sapply(seq(0,1,0.01), function(n) correctRate(unlist(sapply(testingDat
a, function(x) accTest2(x, n)))))
plot(seq(0,1,0.01), threshAccu, type='l')
```




```
max(threshAccu)
```

```
## [1] 0.1037778
```

```
seq(0,1,0.01)[which(threshAccu==max(threshAccu))]
```

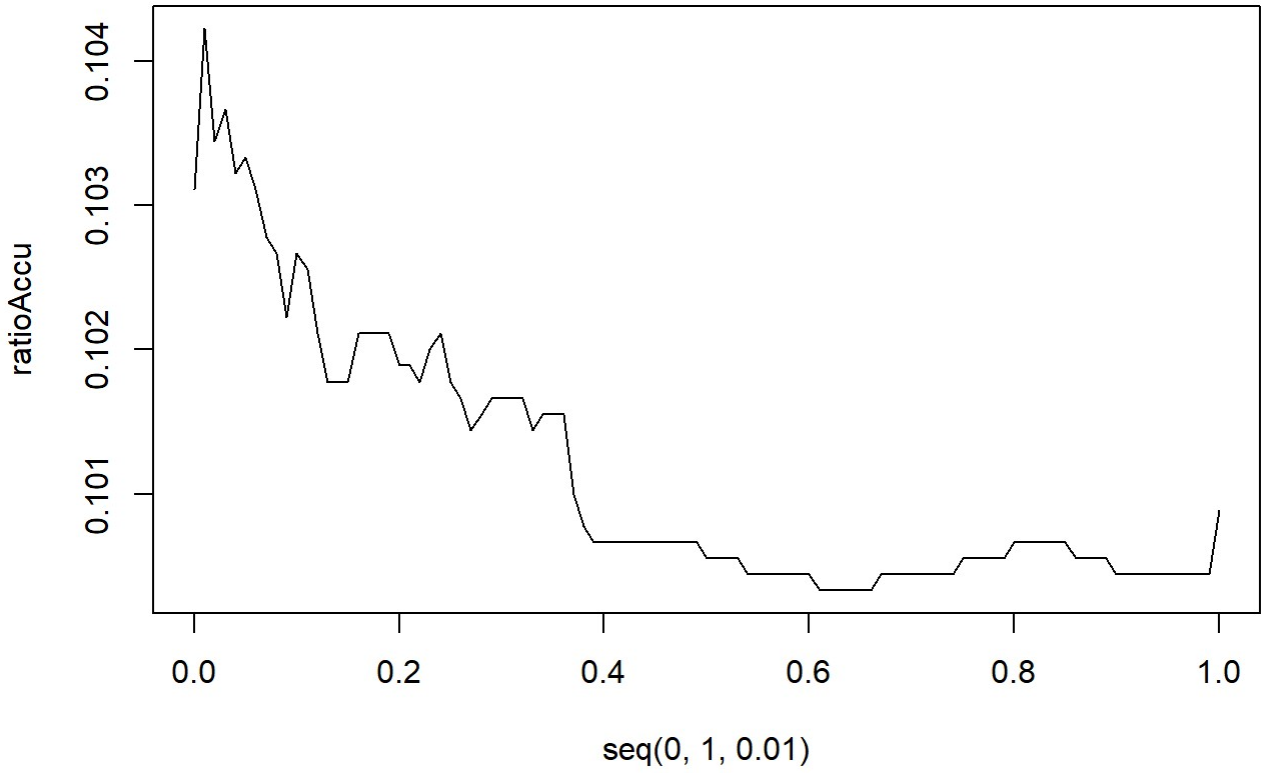
```
## [1] 0.00 0.01 0.02
```

Attempt another modified back-off, this time using ratio of evidence rather than an absolute amount of observations (back off if the n-gram provides less than r times as much evidence as the (n-1)-gram that would be backed off to).

```
backoff3 <- function(x, n) {
  clean <- cleanFunc(x)
  trips <- NGramTokenizer(clean, Weka_control(min=3, max=3))
  pairs <- NGramTokenizer(clean, Weka_control(min=2, max=2))
  if (length(trips) > 0) {
    lastTrip <- trips[length(trips)]
    lastPair <- pairs[length(pairs)]
    if (lastTrip %in% tripData$str & tripData[tripData$str==lastTrip, 'freq'][1, ]/pa
irData[pairData$str==lastPair, 'freq'][1, ] > n) {
      return(as.character(tripData[tripData$str==lastTrip, 'nextWord'][1, ]))
    }
  }
  words <- WordTokenizer(clean)
  if (length(pairs) > 0) {
    lastPair <- pairs[length(pairs)]
    lastWord <- words[length(words)]
    if (lastPair %in% pairData$str & pairData[pairData$str==lastPair, 'freq'][1, ]/wo
rdData[wordData$str==lastWord, 'freq'][1, ] > n) {
      return(as.character(pairData[pairData$str==lastPair, 'nextWord'][1, ]))
    }
  }
  if (length(words) > 0) {
    lastWord <- words[length(words)]
    if (lastWord %in% wordData$str) {
      return(as.character(wordData[wordData$str==lastWord, 'nextWord'][1, ]))
    }
  }
  return(NA)
}
```

Test new new back-off method, scanning across possible values of r to see which provides best results.

```
accTest3 <- function(x, n) sapply(x[1:min(1000, length(x))], function(y) backoff3(str
1Func(y), n)==str2Func(y), USE.NAMES = F)
ratioAccu <- sapply(seq(0,1,0.01), function(r) correctRate(unlist(sapply(testingData,
function(x) accTest3(x, r)))))
plot(seq(0,1,0.01), ratioAccu, type='l')
```




```
max(ratioAccu)
```

```
## [1] 0.1042222
```

```
seq(0,1,0.01)[which(ratioAccu==max(ratioAccu))]
```

```
## [1] 0.01
```

Conclusion

This concludes Tasks 1-5. First, the data was loaded, cleaned, separated into training and test sets, and the training set was tokenized. Next, some basic characteristics of the data such as frequency distribution were explored and plotted. The data was then used to build a model. A prediction model was then implemented, and testing data was tokenized and used to evaluate the prediction model. Finally, two variations on the prediction model were used to further improve the performance of the model.