Week 2

Data Science Capstone

Milestone Report

Completed by Stefan Botha

Outline

This is the week 2 milestone report for the capstone project of the Coursera Data Scienence Specialization. The overal goal of the capstone is to develop a prediction algorithem for the most likely next word in a sequence of words. The purpose of this report is to demonstrate how data was downloaded, imported into R and cleaned. Furthermore it contains some exploratory analyses to investigate some features of the data.

Download and import data

```
setwd("C:/Users/deuscoli/Desktop/Coursera_DataScience/10_Capstone")

if(!file.exists("Coursera-SwiftKey.zip")){

  download.file("https://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-SwiftKey.zip", "Coursera-SwiftKey.zip")

  unzip("Coursera-SwiftKey.zip")

}

blogs <- readLines("final/en_US/en_US.blogs.txt", warn = FALSE, encoding = "UTF-8")

news <- readLines("final/en_US/en_US.news.txt", warn = FALSE, encoding = "UTF-8")

twitter <- readLines("final/en_US/en_US.twitter.txt", warn = FALSE, encoding = "UTF-8")
```

Generate summary stats

Calculate some summary stats for each file: Size in Megabytes, number of entries (rows), total characters and length of longest entry.

```
summary <- data.frame('File' = c("Blogs","News","Twitter"),
```

```
                        "File Size" = sapply(list(blogs, news, twitter), functi
on(x){format(object.size(x),"MB")}),
                        'Nentries' = sapply(list(blogs, news, twitter), functio
n(x){length(x)}),
                        'TotalCharacters' = sapply(list(blogs, news, twitter),
function(x){sum(nchar(x))}),
                        'MaxCharacters' = sapply(list(blogs, news, twitter), fu
nction(x){max(unlist(lapply(x, function(y) nchar(y))))})
                        )
summary
```

```
##       File File.Size Nentries TotalCharacters MaxCharacters
## 1    Blogs  248.5 Mb   899288       206824505         40833
## 2     News   19.2 Mb    77259        15639408          5760
## 3 Twitter  301.4 Mb  2360148       162096031           140
```

Data Cleaning and selection of Corpus

Because the data are so big (see summary table above) we are only going to proceed with a subset (e,g, 5% of each file). Then we are going to clean the data and convert to a corpus.

```
set.seed(1313) # Make subsampling reproducible
sample_size <- 0.05 # Subsamle to 5%
#sample_size <- 0.01 # Subsamle to 1%

blogs_index <- sample(seq_len(length(blogs)),length(blogs)*sample_size)
news_index <- sample(seq_len(length(news)),length(news)*sample_size)
twitter_index <- sample(seq_len(length(twitter)),length(twitter)*sample_size)

blogs_sub <- blogs[blogs_index[]]
news_sub <- news[news_index[]]
twitter_sub <- twitter[twitter_index[]]

library(tm) # Load Text Mining library

# Make corpus out of all 3 sub sampled data sets.
# Then tidy up a bit
```

```
corpus <- Corpus(VectorSource(c(blogs_sub, news_sub, twitter_sub)), readerCon
trol=list(reader=readPlain,language="en")) # Make corpus


# Converting to Corpus seems to mess up the encoding, need to remove non ASCI
I characters

corpus <- Corpus(VectorSource(sapply(corpus, function(row) iconv(row, "latin1
", "ASCII", sub=""")))) # Remove non-ASCII


corpus <- tm_map(corpus, removePunctuation) # Remove punctuation

corpus <- tm_map(corpus, stripWhitespace) # Remove unneccesary white spaces

corpus <- tm_map(corpus, content_transformer(tolower)) # Convert to lowercase

corpus <- tm_map(corpus, removeNumbers) # Remove numbers

corpus <- tm_map(corpus, PlainTextDocument) # Plain text

#corpus <- tm_map(corpus, removeWords, stopwords("english")) # Not sure if I
want to do this
```

N-grams

Now that we have a clean dataset we need to convert it to a format that is most useful for Natural Language Prpcessing (NLP). The format of choice are N-grams stored in Term Document Matrices (TDM). The N-gram representation of a text lists all N-tuples of words that appear. The simplest case is the unigram which is based on individual words. The bigram is based on pairs of to words and so on. The TDMs store the frequencies of the N-grams in the respective sources.

```
library(RWeka) # Weka is a collection of machine learning algorithms for data
mining


# I tried to solve this with a single function that takes 2 arguments instead
of 4 separate functions but could not get it to work.

UnigramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 1, max =
1))

BigramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 2, max =
2))

TrigramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 3, max =
3))

QuadgramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 4, max
= 4))


Unigrams <- TermDocumentMatrix(corpus, control = list(tokenize = UnigramToken
izer))
```

```
Bigrams <- TermDocumentMatrix(corpus, control = list(tokenize = BigramTokeniz
er))

Trigrams <- TermDocumentMatrix(corpus, control = list(tokenize = TrigramToken
izer))

Quadgrams <- TermDocumentMatrix(corpus, control = list(tokenize = QuadgramTok
enizer))


Unigrams

## <<TermDocumentMatrix (terms: 113881, documents: 166833)>>

## Non-/sparse entries: 2342178/18996766695

## Sparsity           : 100%

## Maximal term length: 97

## Weighting          : term frequency (tf)

Bigrams

## <<TermDocumentMatrix (terms: 1112188, documents: 166833)>>

## Non-/sparse entries: 3196975/185546463629

## Sparsity           : 100%

## Maximal term length: 87

## Weighting          : term frequency (tf)

Trigrams

## <<TermDocumentMatrix (terms: 2293274, documents: 166833)>>

## Non-/sparse entries: 3081224/382590700018

## Sparsity           : 100%

## Maximal term length: 97

## Weighting          : term frequency (tf)

Quadgrams

## <<TermDocumentMatrix (terms: 2740230, documents: 166833)>>

## Non-/sparse entries: 2930278/457157861312

## Sparsity           : 100%

## Maximal term length: 103

## Weighting          : term frequency (tf)
```

Exploratory data analysis

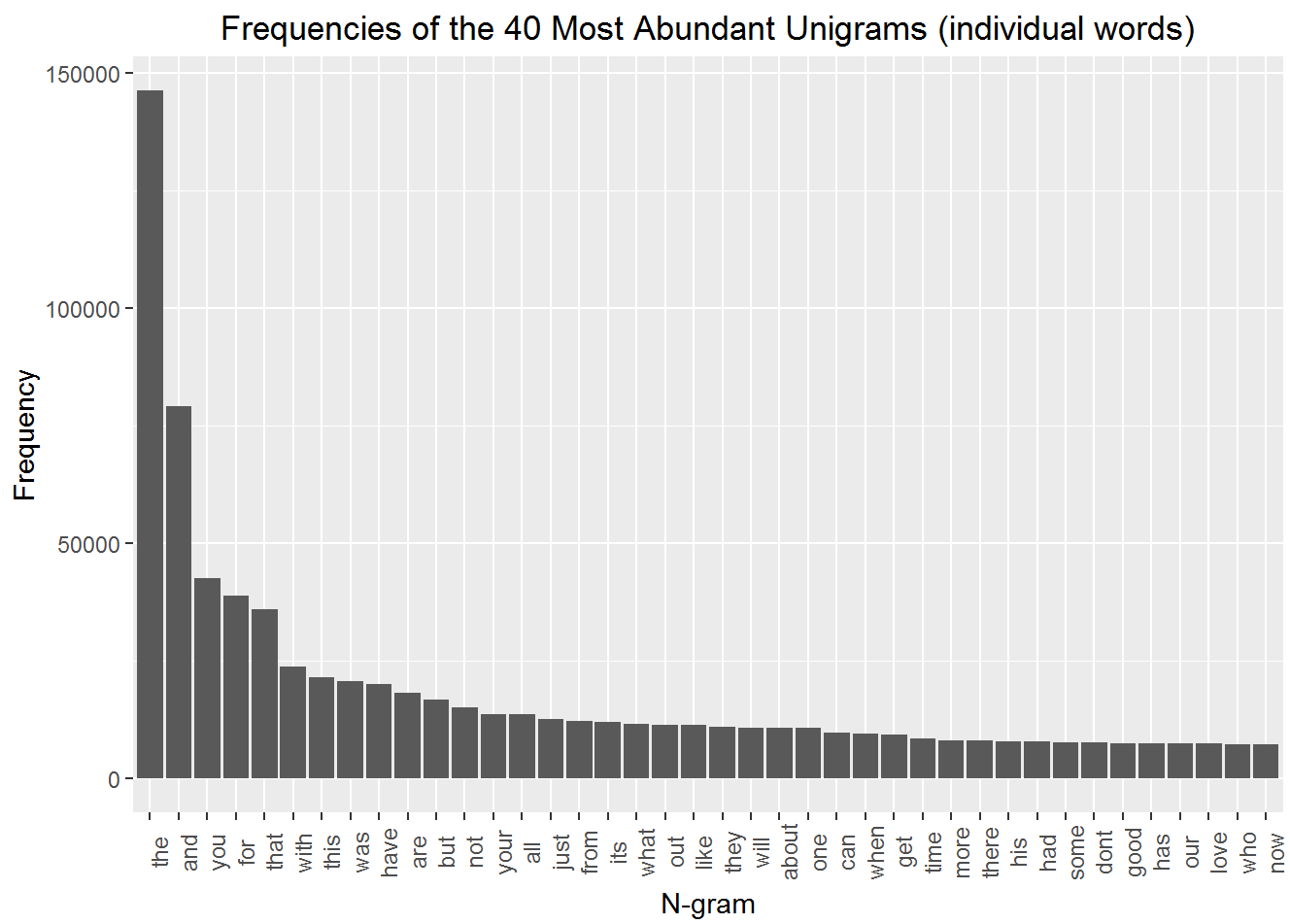
The above matrices are extremely sparse (i.e. they are almost entirely cmposed of zeroes). We need to create a denser matrices to do exploratory analyses and remove rare N-grams.

```
library(ggplot2)


# Function to sum up rows and sort by N-gram frequency
freq_frame <- function(tdm){
    freq <- sort(rowSums(as.matrix(tdm)), decreasing=TRUE)
    freq_frame <- data.frame(word=names(freq), freq=freq)
    return(freq_frame)
}


# Make matrices more dense, add up and sort
UnigramsDense <- removeSparseTerms(Unigrams, 0.999)
UnigramsDenseSorted <- freq_frame(UnigramsDense)
BigramsDense <- removeSparseTerms(Bigrams, 0.999)
BigramsDenseSorted <- freq_frame(BigramsDense)
TrigramsDense <- removeSparseTerms(Trigrams, 0.999)
TrigramsDenseSorted <- freq_frame(TrigramsDense)
QuadgramsDense <- removeSparseTerms(Quadgrams, 0.9999)
QuadgramsDenseSorted <- freq_frame(QuadgramsDense)


GG <- ggplot(data = UnigramsDenseSorted[1:40,], aes(x = reorder(word, -freq),
y = freq)) + geom_bar(stat="identity")
GG <- GG + labs(x = "N-gram", y = "Frequency", title = "Frequencies of the 40
Most Abundant Unigrams (individual words)")
GG <- GG + theme(axis.text.x=element_text(angle=90))
GG
```

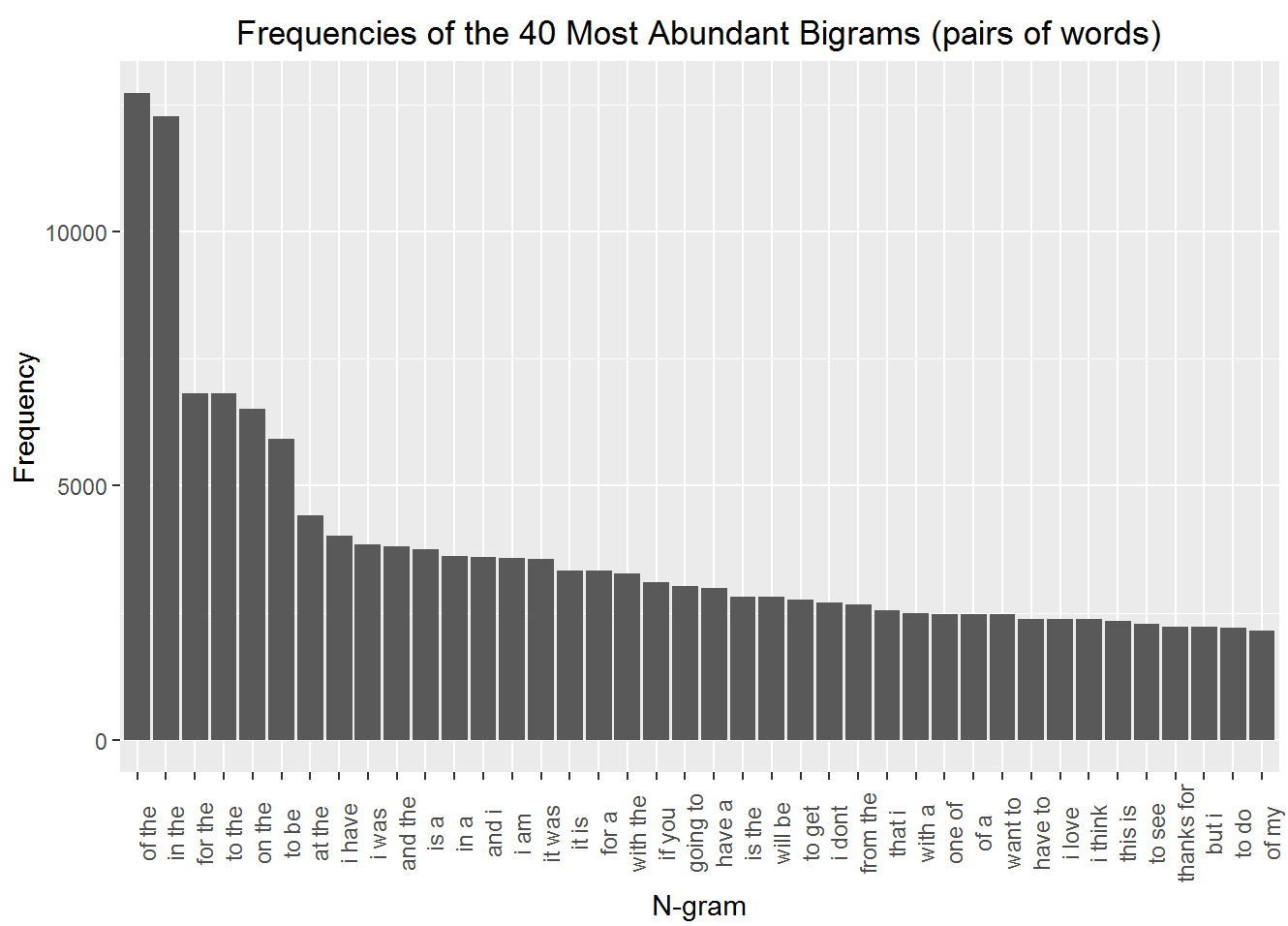



```
GG <- ggplot(data = BigramsDenseSorted[1:40,], aes(x = reorder(word, -freq),
y = freq)) + geom_bar(stat="identity")

GG <- GG + labs(x = "N-gram", y = "Frequency", title = "Frequencies of the 40
Most Abundant Bigrams (pairs of words)")

GG <- GG + theme(axis.text.x=element_text(angle=90))

GG
```

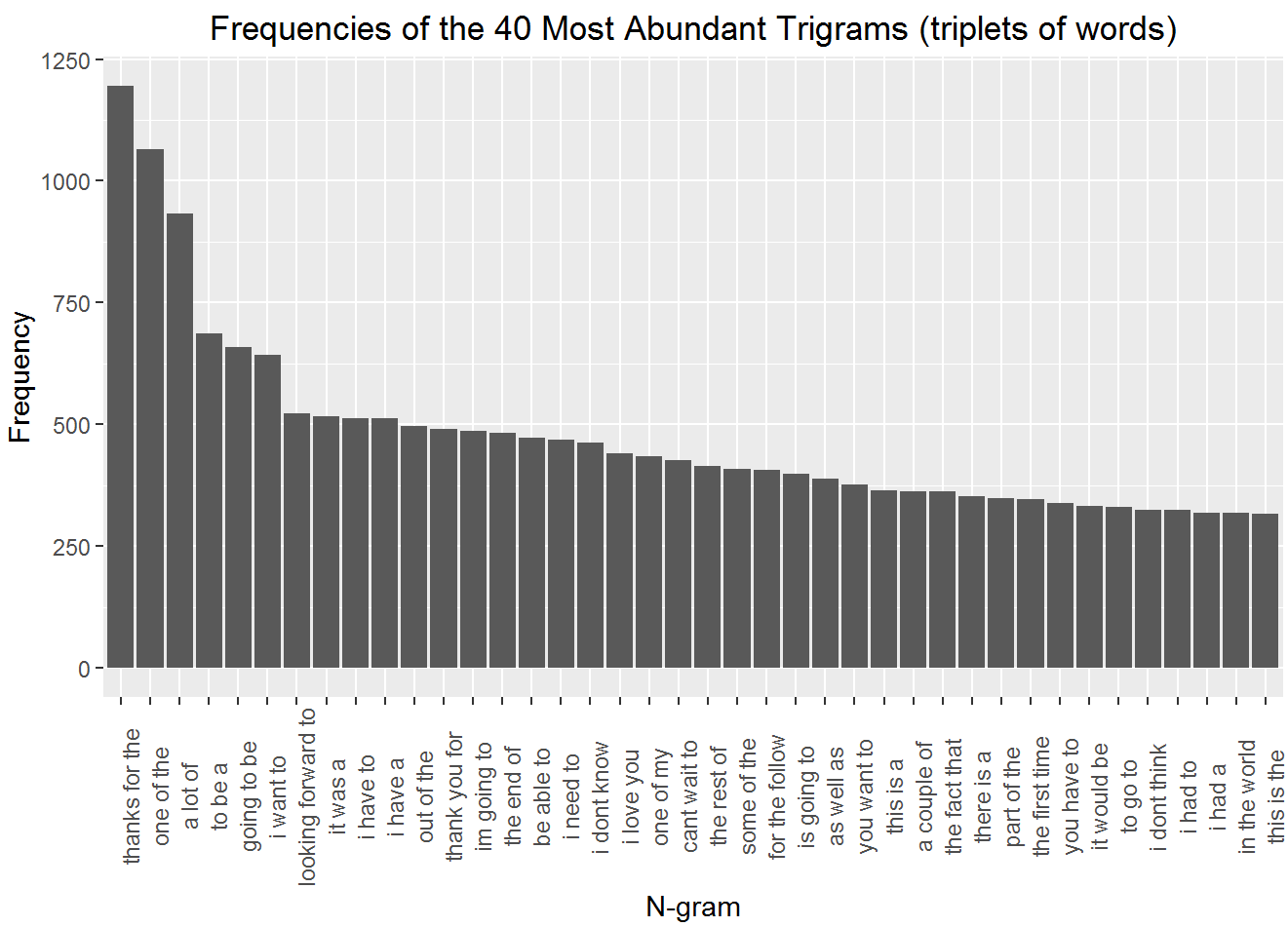



```
GG <- ggplot(data = TrigramsDenseSorted[1:40,], aes(x = reorder(word, -freq),
y = freq)) + geom_bar(stat="identity")

GG <- GG + labs(x = "N-gram", y = "Frequency", title = "Frequencies of the 40
Most Abundant Trigrams (triplets of words)")

GG <- GG + theme(axis.text.x=element_text(angle=90))

GG
```

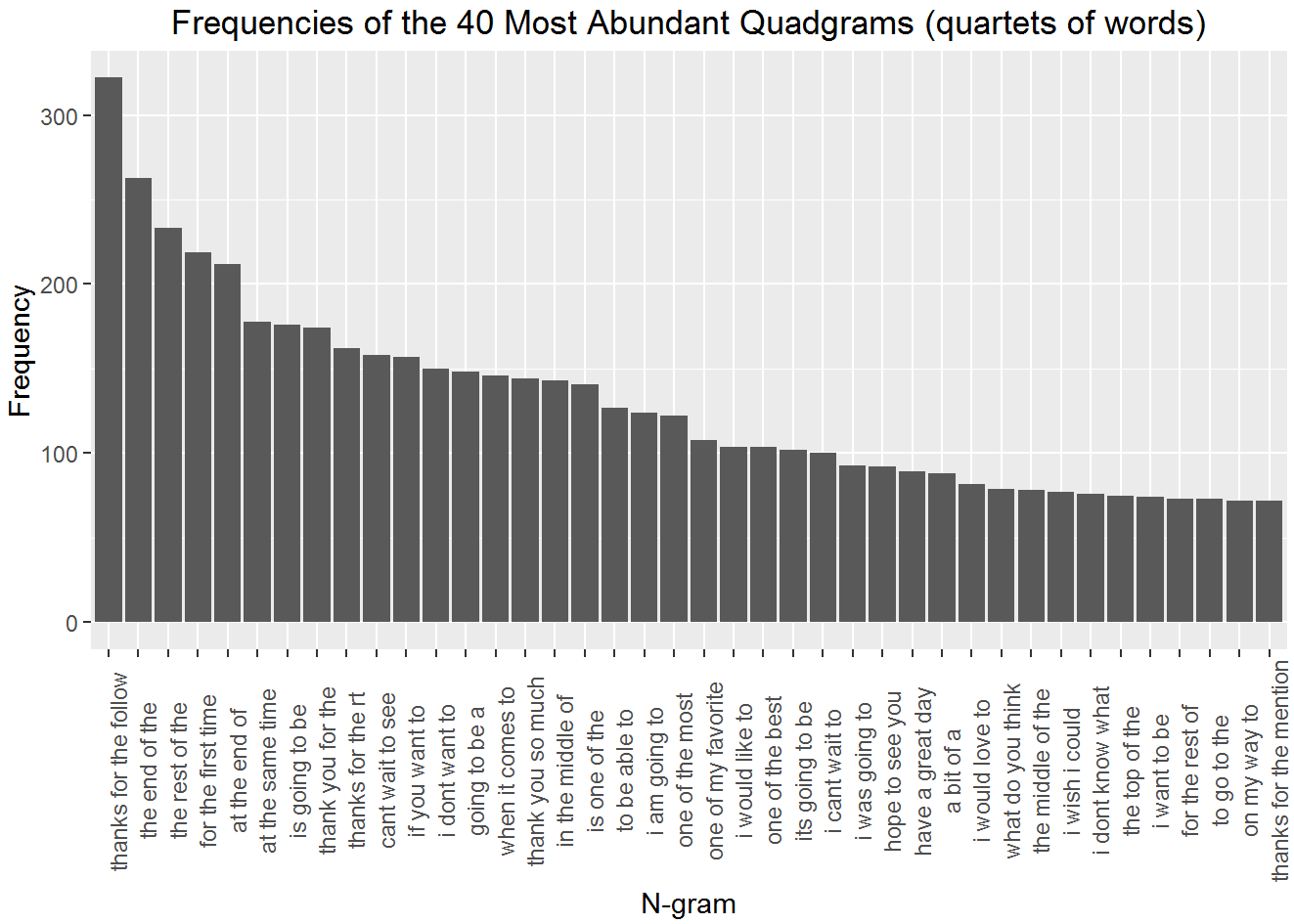



```
GG <- ggplot(data = QuadgramsDenseSorted[1:40,], aes(x = reorder(word, -freq)
, y = freq)) + geom_bar(stat="identity")

GG <- GG + labs(x = "N-gram", y = "Frequency", title = "Frequencies of the 40
Most Abundant Quadgrams (quartets of words)")

GG <- GG + theme(axis.text.x=element_text(angle=90))

GG
```



Summary of Findings

- Building N-grams takes some time, even when downsampling to 5%. Caching helps to speed the process up when run the next time (cache = TRUE).
- The longer the N-grams, the lower their abundance (e.g. the most abundant Trigram's frequency is 1196 and that of the most abundant Quadgrams frequency is 322).

Plans for Prediction Algorithm and Shiny App

The corpus has been converted to N-grams stored in Term Document Matrices and then converted to data frames of frequencies. This format should be useful for predicting the next word in a sequence of words. For example, when looking at a string of 3 words the most likely next word can be guessed by investigating all 4-grams starting with these three words and chosing the most frequent one. The prediction should be fairly quick as the Term Document Matrices are not required for this purpose (it is not important in what document they occur, only their overall frequency is relevant). The data frames of summed up frequencies i.g. QuadgramsDenseSorted should suffice and the intermediate steps can be cached. The Shiny App should be able to run in acceptable time. If not we will need to further reduce the sample size, e.g. from 5% to 1%. This may be not enough data though. Maybe it is worth to even increse the sample size as the time consuming steps only need to run once.