Capstone Plan

Nicole Goebel

November 15, 2014

Summary:

In this report, I explore a corpus of documents from blogs, news, and twitter feeds which were derived from `HC Corpora`. With the ultimate goal of creating a predictive language algorithm, I demonstrate the use of various natural language processing packages in R, such as `tm` and `RWeka` as well as unix shell commands, which were used to clean, analyze and tokenize the corpus as a document-term matrix (i.e., a bag of words) and ngrams. 1% of each document was subsampled, due to their large sizes ranging from 167 Mb for the twitter document to 210 Mb for the blogs document. The corpus of subsamples was then cleaned by converting to lowercase letter and removing special characters, numbers, profanity, odd letters, and whitespace. Analyzed features of the cleaned, subsampled corpus include: 1) the frequency of top 20 words that were an order of magnitude greater than the remaining words, 2) the top 20 words included common words such as 'the', 'and', 'that', 'for', 'you', 'with', etc., 3) when reducing the sparsity of the dataset by 0.1, the frequency of the most frequently used words were not as stark as that of the non-reduced. A variety of outputs, tables and figures are used to demonstrate these features and findings. I also outline my plan for a predictive algorithm to be implemented and published as a Shiny app in R, which involves `backoff` and `interpolation` algorithmic approaches.

Introduction

The ultimate aim of this project is to develop a language prediction system for the company SwiftKey, a leading predictive texting company. In this report, I begin to work with and explore a corpus of text in order to formulate a plan for a language prediction system.

The data used in this project come from a corpus found at HC Corpora. The data have been made available on the course site in a zip file, which contains three files consisting of news, blogs, and tweets. Here, I explain the steps I took in order to transform the corpus of text through cleaning, tokenization and analysis in preparation for formulating a text prediction system.

Data Download and Load

Data were successfully downloaded as a zip file from the course site, loaded into R and unzipped with the following code:

```r
##   a. download file into directory
zipName = 'Coursera-Swiftkey.zip'
if (!file.exists(zipName)){
  fileURL <- "https://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-SwiftKey.zip"
  download.file(fileURL, destfile="Dataset.zip", method="curl")
  filelist <- list.files()
  print(filelist)
  dateDownloaded <- date()
  print(dateDownloaded)
} else {message("Zipfile already exists.")}
## 2. unzip zip file:
zipDir='final'
if (!file.exists(zipDir)) {
  unzip(zipName)
} else {message("Data already unzipped.")}
```

Both terminal and R commands (shown below) could be used to get a sense of the length of the documents in terms of word and line counts.

```
# word (w), line (l), character (m), and byte (c) counts
system("wc -wlmc final/en_US/en_US.blogs.txt")    # blogs document
system("wc -wlmc final/en_US/en_US.news.txt")     # news document
system("wc -wlmc final/en_US/en_US.twitter.txt")  # twitter document
```

```
dirName='~/Dropbox/Courses/Coursera_Data_Science_2014/CapstoneProject/'
#line counts in R
newsLC <- length(readLines(file(paste0(dirName, "/final/en_US/en_US.news.txt"), 'rb', encoding= 'UTF-8'
tweetsLC <- length(readLines(file(paste0(dirName, "/final/en_US/en_US.twitter.txt"), 'rb', encoding= 'U
#gives Warnings for embedded nul in lines 167155, 268547, 1274086, 1759032
blogsLC <- length(readLines(file(paste0(dirName, "/final/en_US/en_US.blogs.txt"), 'rb', encoding= 'UTF-
```

The terminal method was much faster than the R commands shown above, and gave line counts of 2360148 (twitter), 1010242 (news), and 899288 (blogs) txt files. Word counts were for 30374206 (twitter), 34372720 (news), 37334690 (twitter) txt files; these counts reflect the sizes of the twitter (167 Mb), news (206 Mb), and blog (210 Mb) txt files.

There were four warnings given when reading in the twitter file (not shown). I presume this was due to non-printable characters, which were removed (see below).

Subsampling Data

Due to the large size of the files, 1% lines were randomly subsampled from each file (shown in code below) after trimming whitespace at the start and end of each line, and then saved as a new test file for further analysis.

```
# small function for trimming whitespace at start and end of line
trim <- function( x ) {
  gsub("(^[[:space:]]+|[[:space:]]+$)", "", x)
}
# function for randomly subsampling a file
subSampleFileRand <- function(dirNameFull, dirNameFullSub, infile,outfile,perc,header=F) {
  ci <- file(paste0(dirNameFull, infile, ".txt"), "rb", encoding= 'UTF-8')
  co <- file(paste0(dirNameFullSub, outfile, ".txt"), "w")
  y<-system(paste0("wc -l ", dirNameFull, infile, '.txt'), intern=TRUE)
  numLines<-as.numeric(strsplit(trim(y)," ")[[1]][1])
  m<-round(numLines*perc) #get a % of lines
  cullrecs <- sort(sample(1:numLines,m,replace=FALSE))
  if (header) {
    hdr <- readLines(ci,n=1)
    writeLines(hdr,co)
  }
  recnum = 0
  numout = 0
  while (TRUE) {
    inrec <- readLines(ci,n=1)
    if (length(inrec) == 0) { # end of file?
      close(co)
      #return(numout)
    }
    recnum <- recnum + 1
    if (recnum %in% cullrecs) {
      numout <- numout + 1
      writeLines(inrec,co)
    }
  }
}
# use function above to subsample each file
dirNameFull='~/Dropbox/Courses/Coursera_Data_Science_2014/CapstoneProject/final/en_US/'
dirNameFullSub='~/Dropbox/Courses/Coursera_Data_Science_2014/CapstoneProject/final/en_U
/subsample/'
suffixes=c("news", "twitter", "blogs")
if(sum(file.exists(suffixes)) < 3) {
  print(paste0("subsampling each file"))
  k=0.01 #% of file sampled
  subSampleFileRand(dirNameFull, dirNameFullSub, "en_US.twitter","twitterSubRand",k,header=F)
  closeAllConnections()
  subSampleFileRand(dirNameFull, dirNameFullSub, "en_US.blogs","blogsSubRand",k,header=F)
  closeAllConnections()
  subSampleFileRand(dirNameFull, dirNameFullSub, "en_US.news","newsSubRand",k,header=F)
  closeAllConnections()
}
```

Before compiling txt files using the `R` package `tm`, the files were processed with the follwing command in order to remove non-printable characters:

```
tr -c "[[:alnum:][:space:]\r" " " < twitterSubRand.txt >twitterSubRandClean.txt
tr -c "[[:alnum:][:space:]\r" " " < newsSubRand.txt >newsSubRandClean.txt
tr -c "[[:alnum:][:space:]\r" " " < blogsSubRand.txt >blogsSubRandClean.txt
```

Once the files were subsampled, these reduced files were compiled into an object, called `corpus`, that could be further manipulated (i.e., cleaned and analyzed). This step was carried out using the `R` text mining package `tm`.

```
dirNameFullSub='~/Dropbox/Courses/Coursera_Data_Science_2014/CapstoneProject/final/en_US/subsample/'
files <- DirSource(directory=dirNameFullSub, encoding='latin1')
corpus <- VCorpus(x=files)
summary(corpus)
```

```
##                         Length Class             Mode
## blogsSubRandClean.txt   2      PlainTextDocument list
## newsSubRandClean.txt    2      PlainTextDocument list
## twitterSubRandClean.txt 2      PlainTextDocument list
```

Cleaning Data

The next step was to clean the text data (i.e., remove unnecessary symbols, whitespace, etc). I cleaned the data by removing punctuation, numbers, symbols and whitespace. I also converted all letters to lowercase. These transformations, as well as examples of how to inspect the corpus, are shown in the following code:

```
toSpace <- content_transformer(function(x, pattern) gsub(pattern, " ", x))
corpus <- tm_map(corpus, toSpace, "[^0-9A-Za-z// ]") #removes special characters
corpus <- tm_map(corpus, toSpace, "/|@|\\|")
corpus <- tm_map(corpus , removeNumbers)
badWords <- c("shit", "fuck", "ass", "bitch")
corpus <- tm_map(corpus,removeWords,badWords)
#corpus <- tm_map(corpus, removePunctuation)
#corpus <- tm_map(corpus, removeWords, stopwords("english"))
corpus <- tm_map(corpus, removeWords, c("s", "d", "m", "t")) #remove odd letters
corpus <- tm_map(corpus, stripWhitespace)
corpus <- tm_map(corpus, tolower)
corpus <- Corpus(VectorSource(corpus)) #corrects format after converting to lower case
#summary(corpus)
corpus[[1]]$meta   #meta data
```

```
## Metadata:
##   author       : character(0)
##   datetimestamp: 2014-11-16 17:46:18
##   description  : character(0)
##   heading      : character(0)
##   id           : 1
##   language     : en
##   origin       : character(0)
```

```
length(corpus[[1]]$content) # get document 1 length
```

```
## [1] 8993
```

```
corpus[[1]]$content[1]      # look at line 1 of first document in corpus
```

```
## [1] "the juices today were the best yet breakfast was like a chilled spiced carrot soup and lunch wa
```

Cleaning text data was necssary to pare down the text to the best set of features (words) needed in order to predict the next word. This step is important for optimizing performance and accuracy of the word bank used to formulate a predictive text model.

Tokenization and Exploratory Analyses

Tokenization is a way to explain to a computer how sentences are broken down into words and how sentences are formed. With a cleaned and tokenized data set, the tokens can be converted to a document-term matrix, which can then be used for predictive modeling, i.e., a machine learning algorithm can be used to train a model and predict words; the correct feature can be selected by choosing the one that scores more favorably (e.g., a lower p-value or higher probability).

The text data were tokenized (i.e., divided text into words) by creating a document-term matrix. The document-term matrix enables a view of the frequency of terms as unigrams, also known as a bag of words, across the three document subsamples. In the code below, the document-term matrix is created, inspected and analyzed (see figures); as shown, terms can be searched for by index or by word. The sparsity and maximal term length for the document term matrix are printed out below.

```r
#create document term matrix (dtm) - discard words shorter than length of 2
dtm <- DocumentTermMatrix(corpus, control = list(minWordLength = 2))
tdm <- TermDocumentMatrix(corpus, control = list(minWordLength = 2))
dtmDim<-dim(dtm)
dtm  #information about document term matrix
```

```
## <<DocumentTermMatrix (documents: 3, terms: 48790)>>
## Non-/sparse entries: 77470/68900
## Sparsity           : 47%
## Maximal term length: 91
## Weighting          : term frequency (tf)
```

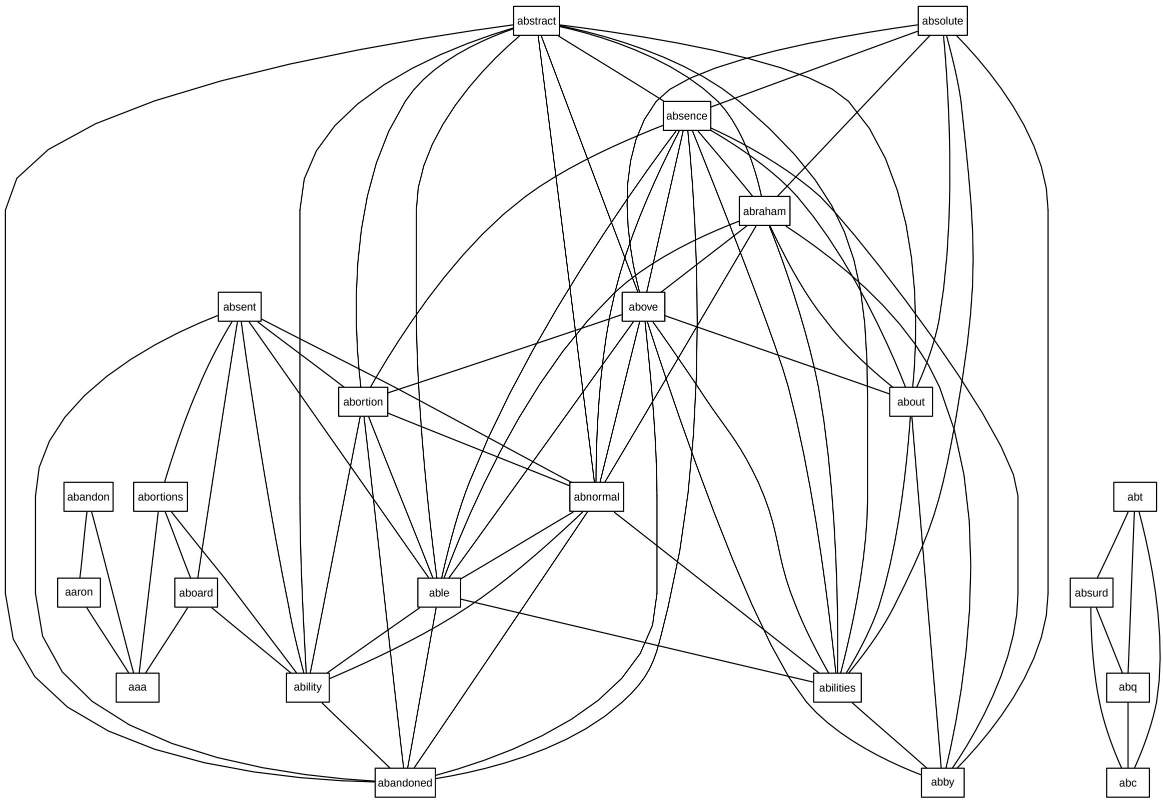
```r
inspect(dtm[1:3, 5110:5120])        #inspect a part of the dtm by index
```

```
## <<DocumentTermMatrix (documents: 3, terms: 11)>>
## Non-/sparse entries: 16/17
## Sparsity           : 52%
## Maximal term length: 13
## Weighting          : term frequency (tf)
##
##     Terms
## Docs braindead brained brainer brainery brains brainstem brainstorm
##    1         0       1       3        1      2         0          0
##    2         0       0       1        0      2         1          0
##    3         1       0       0        0      8         0          1
##     Terms
## Docs brainstorming brainwashed brainy braised
##    1             1           0      1       0
##    2             0           1      1       3
##    3             1           0      0       0
```

```r
inspect(dtm[1:3, c("offense", "still", "struggling", "but", "the", "referees", "crowd", "players", "def
```

```
## <<DocumentTermMatrix (documents: 3, terms: 9)>>
## Non-/sparse entries: 26/1
## Sparsity           : 4%
## Maximal term length: 10
## Weighting          : term frequency (tf)
##
##     Terms
## Docs offense still struggling  but   the referees crowd players defense
##    1       3   371         17 2027 19002        0    20      23      20
##    2      30   285         22 1451 20029        2    44     106      82
##    3      14   370          5 1258  9426        1    15      32      10
```

```r
plot(dtm, terms = findFreqTerms(tdm, lowfreq = 6)[1:25], corThreshold = 0.75, weighting=FALSE)
```



matrix for 1-grams-1.pdf

Figure 1. Correlations on a selection of terms with a minimum frequency of 6 within the document-term matrix, implementing a correlation threshold of 0.75.

The document-term matrix was then explored in order to look for the number and frequency of terms in the subsampled documents. A plot of the frequency per word shown in the figure below indicates that the top 15 to 20 words were used most often (a frequency of 1000 to 19000) and the remaining words were used at a similar, low frequency (on the order of 100 or less). This large discrepancy is due to the most frequent words being very common (e.g., 'the', 'and', 'that', 'for', 'you', 'with', etc).

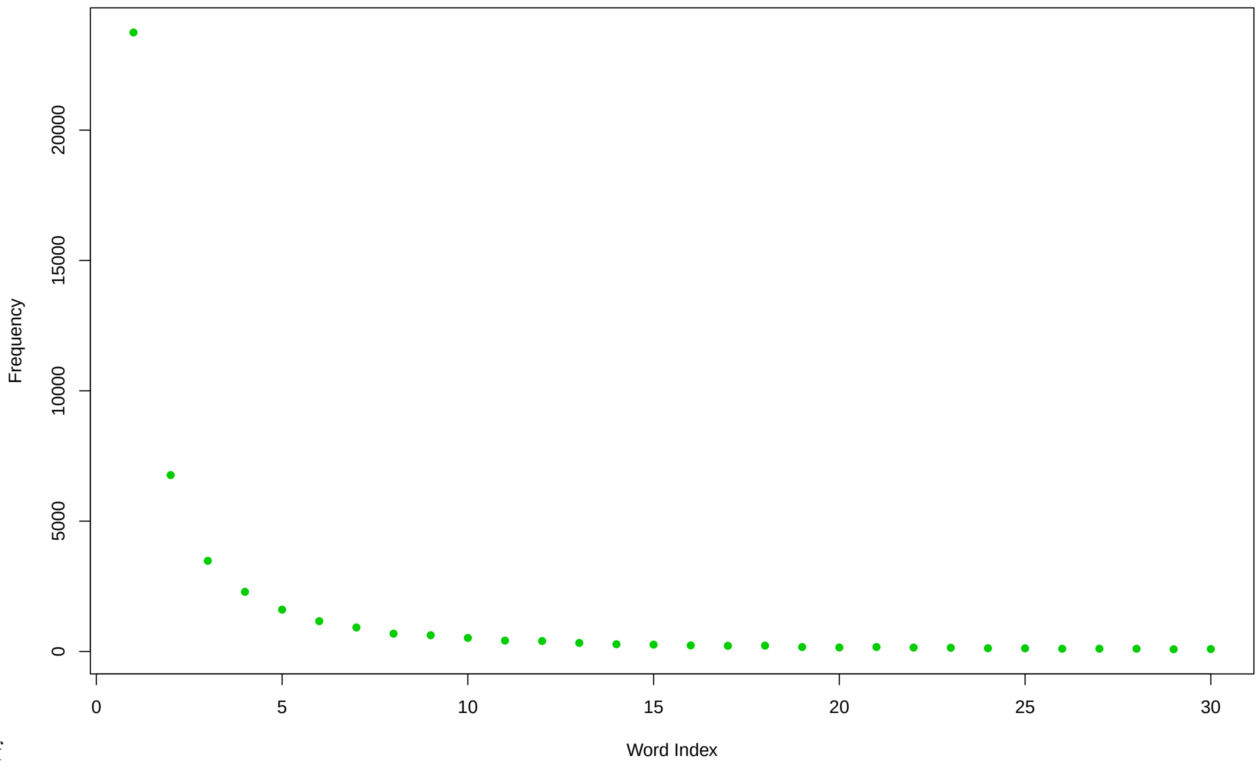
```
#obtain frequency of terms across all documents in corpus (sum across columns in a matrix)
freqo <- colSums(as.matrix(dtm)) #original
freq <- sort(colSums(as.matrix(dtm)), decreasing=TRUE) #decreasing
length(freq)
```

```
## [1] 48790
```

```
head(table(freq), 30)
```

```
## freq
##     1     2     3     4     5     6     7     8     9    10    11    12
## 23745  6766  3478  2288  1606  1163   922   684   624   521   418   402
##    13    14    15    16    17    18    19    20    21    22    23    24
##   328   278   263   233   218   225   170   155   172   151   143   125
##    25    26    27    28    29    30
##   120   105   107   105    86    94
```

```r
plot(head(table(freq), 30), ylab="Frequency", xlab="Word Index", type='p', pch=16, col=3)
```

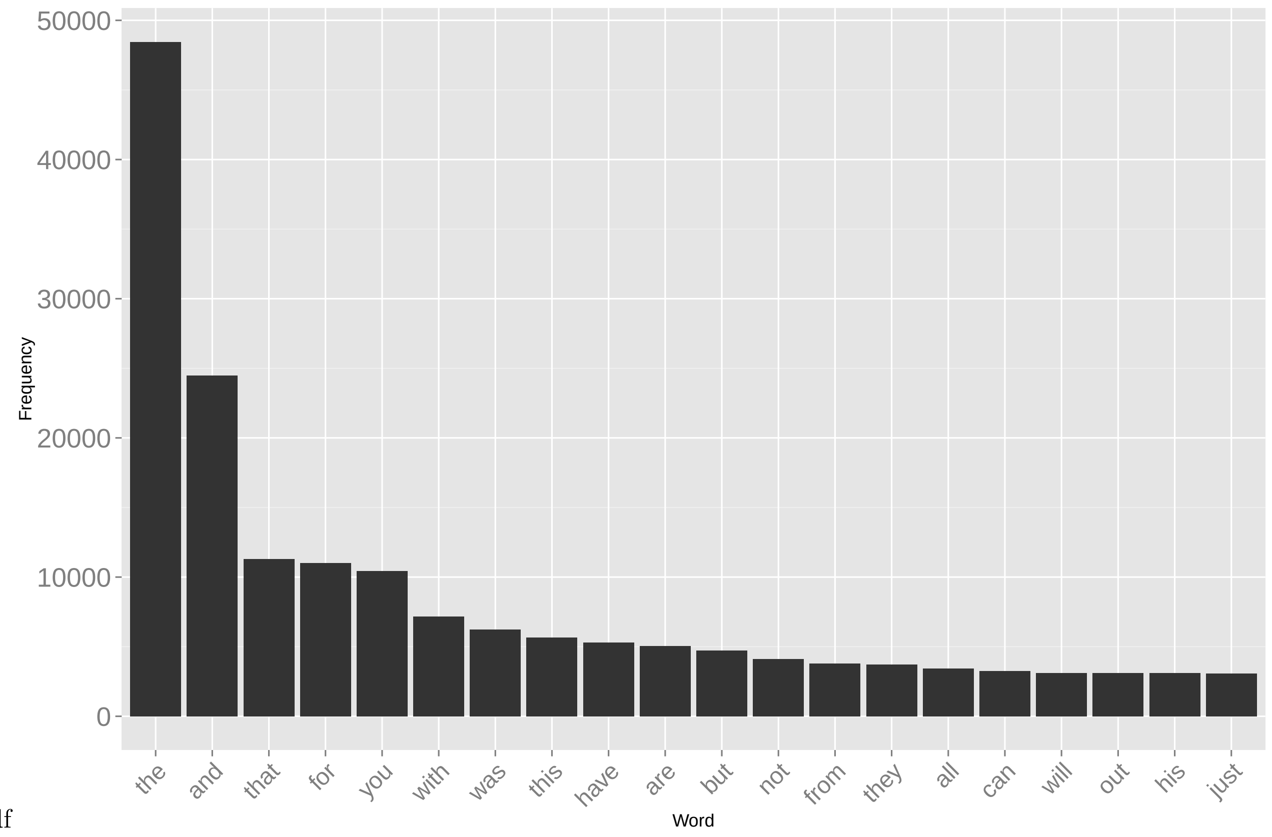


and order-1.pdf

Figure 2. The frequency of words used in the corpus levels off at around 15 to 20 words. The top 5 words are used more frequently than the remaining words by two to three orders of magnitude.

The frequency count of actual words in the corpus is calculated in the code below and shown Figure 3.

```r
wf <- data.frame(word=names(freq), freq=freq) #put freq into data frame for plot
wfo<-wf[order(-wf$freq),]  #order by descending frequencies
wfo$word <- factor(wfo$word, levels = wfo$word[order(-wfo$freq)])
ggplot(wfo[1:20,], aes(word, freq)) + geom_bar(stat="identity") +
  xlab('Word') + ylab('Frequency') +
  theme(axis.text.x=element_text(angle=45, hjust=1, size=16),
        axis.text=element_text(size=18))
```



freq-1.pdf

Figure 3. Frequency of the top 20 words in `corpus` subsample. The majority of the top 20 words are common words such as 'the', 'and', etc.

An attempt to remove sparse terms was carried out below. The removeSparseTerms() function relies on a sparsity parameter which is maximal at 0 and minimal at 1 (i.e., a larger sparsity parameter allows for a larger number of terms, or more sparseness). To test this function, I implemented a sparseness of 0.1 in the code below. Figure 4 compares the frequency of words in the original and the reduced document-term matrix.

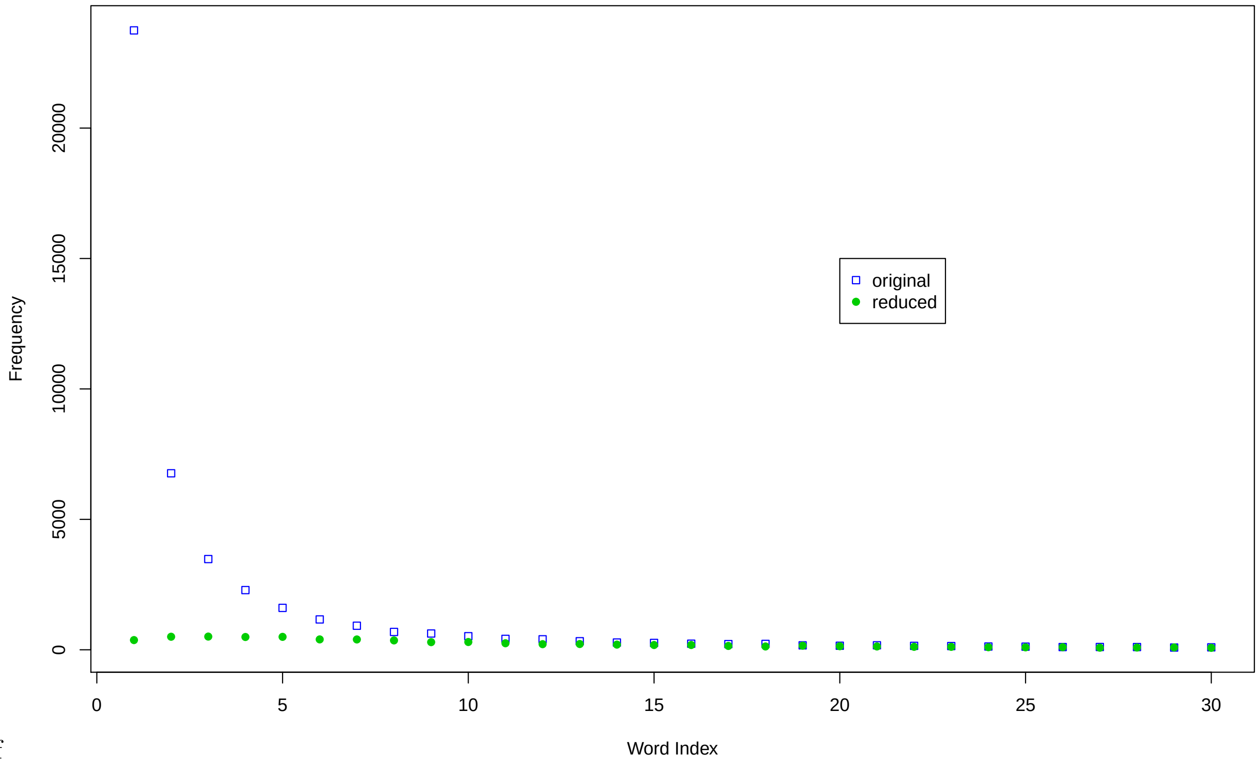
```
dtmS <- removeSparseTerms(dtm, 0.1)
dtmSDim<-dim(dtmS)
freqS <- colSums(as.matrix(dtmS)) #frequency of terms across all documents in corpus
#order frequencies to list most and least frequent terms
ordS <- order(freqS, decreasing=TRUE)
freqS[head(ordS, 10)] # top most frequent words in reduced dtm
```

```
##   the   and  that   for   you  with   was  this  have   are
## 48457 24493 11301 11004 10439  7161  6220  5674  5290  5042
```

```
freqS[tail(ordS, 10)] # least most frequent words in reduced dtm
```

```
##  waterfall     waving  weinstein     whedon      whine   whispers
##          3          3          3          3          3          3
##        wig windshield   wordplay wrongdoing
##          3          3          3          3
```

```r
#head(table(freqS), 30)
plot(head(table(freq), 30), ylab="Frequency", xlab="Word Index", type='p', pch=22, col=4)
points(head(table(freqS), 30), pch=16, col=3)
legend(20, 15000, c('original', 'reduced'), lty=c(-1, -1), pch=c(22, 16), col=c(4, 3))
```



removal-1.pdf

Figure 4. The frequency of words used in the reduced document-term matrix is compared to that in the original. The frequency of the most commonly used words is greatly reduced in the reduced document-term matrix.

This sparseness manipulation reduced the previous 48790 terms to 9778 number of terms. In addition, the frequency of the top words were reduced by at least an order of magnitude.

Frequent terms and associations can be determined with the document-term matrix. The frequency of terms is investigated below, by determining which words had a frequency above the `lowfreq` threshold. For a `lowfreq` threshold of 1000, the reduced document term matrix did not differ from the original.

```r
findFreqTerms(dtm, lowfreq=500) #frequency of terms for original dtm
```

```
##   [1] "about"     "after"     "again"     "all"       "also"
##   [6] "always"    "and"       "another"   "any"       "are"
##  [11] "around"    "back"      "because"   "been"      "before"
##  [16] "being"     "best"      "better"    "big"       "both"
##  [21] "but"       "can"       "city"      "come"      "could"
##  [26] "day"       "days"      "did"       "didn"      "does"
##  [31] "doing"     "don"       "done"      "down"      "each"
##  [36] "end"       "even"      "ever"      "every"     "family"
##  [41] "feel"      "few"       "find"      "first"     "follow"
##  [46] "for"       "from"      "game"      "get"       "getting"
##  [51] "give"      "going"     "good"      "got"       "great"
##  [56] "had"       "happy"     "has"       "have"      "help"
##  [61] "her"       "here"      "him"       "his"       "home"
##  [66] "hope"      "how"       "into"      "its"       "just"
##  [71] "keep"      "know"      "last"      "let"       "life"
##  [76] "like"      "little"    "lol"       "long"      "look"
##  [81] "looking"   "lot"       "love"      "made"      "make"
##  [86] "man"       "many"      "may"       "more"      "most"
##  [91] "much"      "need"      "never"     "new"       "next"
##  [96] "night"     "not"       "now"       "off"       "old"
## [101] "one"       "only"      "other"     "our"       "out"
## [106] "over"      "own"       "part"      "people"    "place"
## [111] "play"      "put"       "really"    "right"     "said"
## [116] "same"      "say"       "says"      "school"    "see"
## [121] "she"       "should"    "show"      "since"     "some"
## [126] "something" "state"     "still"     "such"      "sure"
## [131] "take"      "team"      "than"      "thank"     "thanks"
## [136] "that"      "the"       "their"     "them"      "then"
## [141] "there"     "these"     "they"      "thing"     "things"
## [146] "think"     "this"      "those"     "three"     "through"
## [151] "time"      "today"     "too"       "two"       "use"
## [156] "very"      "want"      "was"       "way"       "week"
## [161] "well"      "were"      "what"      "when"      "where"
## [166] "which"     "while"     "who"       "why"       "will"
## [171] "with"      "work"      "world"     "would"     "year"
## [176] "years"     "you"       "your"
```

```r
findFreqTerms(dtmS, lowfreq=500) #frequency of terms for reduced dtm
```

```
##   [1] "about"     "after"     "again"     "all"       "also"
##   [6] "always"    "and"       "another"   "any"       "are"
##  [11] "around"    "back"      "because"   "been"      "before"
##  [16] "being"     "best"      "better"    "big"       "both"
##  [21] "but"       "can"       "city"      "come"      "could"
##  [26] "day"       "days"      "did"       "didn"      "does"
##  [31] "doing"     "don"       "done"      "down"      "each"
##  [36] "end"       "even"      "ever"      "every"     "family"
##  [41] "feel"      "few"       "find"      "first"     "follow"
##  [46] "for"       "from"      "game"      "get"       "getting"
##  [51] "give"      "going"     "good"      "got"       "great"
##  [56] "had"       "happy"     "has"       "have"      "help"
##  [61] "her"       "here"      "him"       "his"       "home"
##  [66] "hope"      "how"       "into"      "its"       "just"
##  [71] "keep"      "know"      "last"      "let"       "life"
##  [76] "like"      "little"    "long"      "look"      "looking"
##  [81] "lot"       "love"      "made"      "make"      "man"
##  [86] "many"      "may"       "more"      "most"      "much"
##  [91] "need"      "never"     "new"       "next"      "night"
##  [96] "not"       "now"       "off"       "old"       "one"
## [101] "only"      "other"     "our"       "out"       "over"
## [106] "own"       "part"      "people"    "place"     "play"
## [111] "put"       "really"    "right"     "said"      "same"
## [116] "say"       "says"      "school"    "see"       "she"
## [121] "should"    "show"      "since"     "some"      "something"
## [126] "state"     "still"     "such"      "sure"      "take"
## [131] "team"      "than"      "thank"     "thanks"    "that"
## [136] "the"       "their"     "them"      "then"      "there"
## [141] "these"     "they"      "thing"     "things"    "think"
## [146] "this"      "those"     "three"     "through"   "time"
## [151] "today"     "too"       "two"       "use"       "very"
## [156] "want"      "was"       "way"       "week"      "well"
## [161] "were"      "what"      "when"      "where"     "which"
## [166] "while"     "who"       "why"       "will"      "with"
## [171] "work"      "world"     "would"     "year"      "years"
## [176] "you"       "your"
```

Associations to a specific word, within a specified correlation limit (i.e., how closely associated the words are desired to be where a value of 1 indicates an association 100% of the time.) was calculated. Below, associations with the words 'but', 'the' and 'defense' were investigated. This association function was used as a bigram model to predict a subsequent word and was used for Quiz 2 (along with the terminal shell commands `egrep` and `wc` which could investigate the full corpus more efficiently when there was a tie among predicted words using the submsampled corpus.).

```
tmp<-findAssocs(tdm, c("but", "the", "defense"), c(0.75, 0.75, 0.75))
tmp$the[which(names(tmp$the)=="struggling")]  #find association between 'struggling' and 'the'
```

```
## struggling
##       0.98
```

While unigrams can be useful for predicting associations, bi-, tri- and quad-grams can produce an even better model by looking for specific word sequences rather than just word associations. Below, bi- and tri-grams are derived from the corpus. These ngrams can then be searched manually for the frequency of specific word sequences as shown in the examples below. This is a precursor to the predictive model that I intend to implement.

```
ngrams<-NGramTokenizer(corpus, Weka_control(min = 2, max = 3, delimiters = " \\r\\n\\t"))
ngramsTable<-table(ngrams)
# search ngrams for the frequency of specific word sequences:
ngramsTable[which(names(ngramsTable)=="case of")]  # bigram count
```

```
## case of
##      27
```

A clustering algorithm was carried out in order to investigate how the three documents (1=blogs, 2=news, 3=twitter) differed from one another. As shown in the figure below, blogs and twitter feeds are more closely related to one another than the news, which makes sense. Therefore, if one is to test the ability of a language model to predict subsequent words, it would make sense to avoid using a mix of news text with blogs and/or twitter feeds (otherwise you might get very poor accuracy when testing your prediction algorithm).

```
#use dtmS which has removed sparse terms to cluster words
m2 <- as.matrix(dtm)   #create matrix
distMatrix <- dist(scale(m2))  #calculate distance between terms
fit <- hclust(distMatrix, method="ward.D") #cluster terms using ward agglomeration method
plot(fit)
```

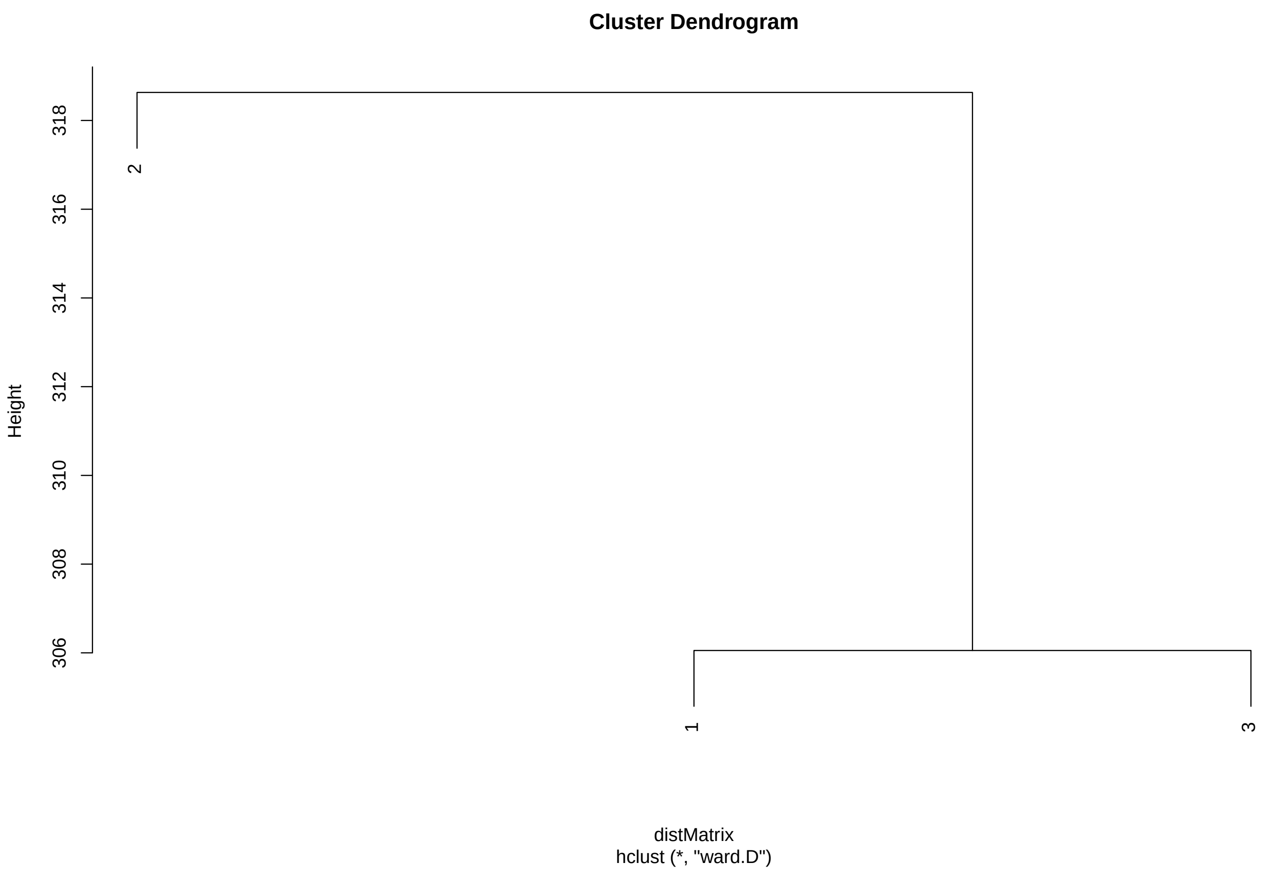
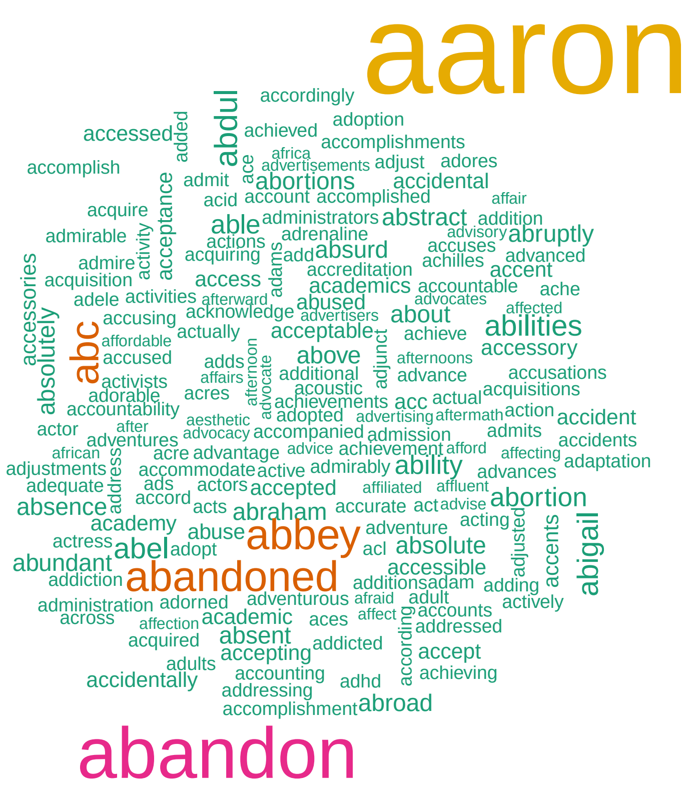


Figure 5. A clustering algorithm to the news (2), blogs (1), and twitter (3) documents indicates that the blogs and twitter documents are more similar to one another than the news document.

Word Cloud

For a fun vizualization, the document-term matrix can be used to build a word cloud.

```
wordcloud(names(freqS), freq, min.freq=500, colors=brewer.pal(6, "Dark2"))
```



cloud-1.pdf

Figure 6. Word cloud for words with a frequency greater than 500.

Plans for predictive algorithm and shiny app

Several approaches can be taken to create a predictive language model. The most simplifying approach is the `Markov model`, which calculates the probablity of a word given another word, as a product of the individual words (i.e., count the number of times both words occur in the same line and divide by the number of times the known word occurs. The association with the highest probability - or score - is the most likely prediction). With this modeling approach, a unigram model that predicts the association of one word with another (in no particular order) is the most basic, but because it assumes that words are independent, it is the least accurate. Even more accurate is a bigram model, which conditions a word on the single previous word: simply count the number of times a word-pair occurs and divide by the number of the first word in the pair in order to get a probability. The highest calculated probability is likely to be the best prediction. A similar approach can be made with tri- and quad-grams. While increasing the number of previous words can improve accuracy of the predicted word, one may ultimately need to consider the subject of the sentence when considering long-distance dependencies (i.e., if the subject of the sentence is many words away from the location where one is trying to predict the word, more context is needed for prediction than simply the 3 to 4 previous words.). Requiring context gets into sentiment analysis, which may be beyond the scope of an initial predictive model.

Unigram, bigram and trigrams can be used with the `backoff` modeling approach. This method estimates an ngram by considering progressively shorter histories, as demonstrated here by an instance where one is predicting the fourth word of a quadgram: the ngram matrix is searched for the most likely quadgram (i.e., highest conditional probability), and if none exists searches for the most likely trigram, and if none exists searches for a bigram, and if none exists utilizes the unigram. I carried out a manual version of this method to answer the quiz 2 questions. A major issue with this method is the erratic change in probability estimates when adding more data due to selection of a different order of ngram on which to base the prediction.

An even better approach than the `backoff` method is to use a combination of uni-, bi- and tri-grams in a `linear interpolation` model. By adding up weighted probabilities for the trigram, bigram and unigram, one can calculate the most likely predicted trigram. This method is complicated by choosing the weights for each ngram, which can be set on a 'hold-out' corpus (another set of data that differs from the training and test sets). Another complication is dealing with unseen words; the probability of these words can be approximated by labelling them as as '' and treated as if they were the same word. While this may be the most accurate method, it could be too wieldy for web-scale ngrams.

The `stupid backoff` smoothing method can be also used to increase efficiency of web-scale ngrams. With this method, scores rather than probabilities are produced in a similar fashion to the `backoff` method: a count of words in the longest ngram divided by the count of the previous word is used to calculate a score as long as the count of the previous word is greater than zero, if this is not the case, a weighted score using the second-most previous word is used so long as this value is not zero. This continues until finally a unigram count is used as a last resort (i.e., the count of the previous word divided by the total number of words).

For the purpose of simplicity and ability to calculate large ngrams, I plan to use the uni-, bi- and tri-grams created above to implement the `stupid backoff` smoothing algorithm described above. If poor accuracy is obtained using this model for a range in parameter weights, I will attempt to implement the more elaborate interpolation method. Once an algorithm is working reliably, I will implement a Shiny app in `R` that attempts to predict the the most likely next word in two to four word ngrams.

When attempting the above approaches, I will have to 1) calculate quad-grams, 2) increase the size of my subsample in order to capture variability, 3) possibly avoid using the news file when training and testing the algorithm since it appears to be very different from the blogs and twitter files. I will also attempt to use various levels of sparseness reduction in order to tune the speed and accuracy of the predictive algorithm.