Data Science Capstone - Milestone Project

Steve Johnson

December 6, 2018

Background

The goal of this report is to demonstrate that I have acquired and performed exploratory analysis of the training data that will eventually be used to create a predictive text algorithm. This data consists of English text from 3 different sources: Twitter, Blogs, and News. Since biases in training data will have significant impact on algorithm performance, it will be important for the final project to uncover major biases in the structure and makeup of these data sets.

Data Acquisition

Initial attempt to load the data revealed premature end of file characters in the news data file. These were removed by hand using Windows Notepad find and replace functionality.

```r
# Open file connection to twitter dataset
con1 <- file("C:/Users/Steve/Documents/RSaves/Capstone/Coursera-
SwiftKey/final/en_US/en_US.twitter.txt", "r")
# Open file connection to blogs dataset
con2 <- file("C:/Users/Steve/Documents/RSaves/Capstone/Coursera-
SwiftKey/final/en_US/en_US.blogs.txt", "r")
# Open file connection to news dataset
con3 <- file("C:/Users/Steve/Documents/RSaves/Capstone/Coursera-
SwiftKey/final/en_US/en_US.news.txt", "r")

data_twit<-readLines(con1,-1)
close(con1)
# 2,360,141 lines of data in twitter file. Each line a character vector
data_blog<-readLines(con2,-1)
close(con2)
# 899,286 lines of data in blogs file. Each line a character vector
data_news<-readLines(con3,-1)
close(con3)
# Generate data for raw data summary table.
wcT<-wordcount(paste(data_twit, collapse=''))
wcB<-wordcount(paste(data_blog, collapse=''))
wcN<-wordcount(paste(data_news, collapse=''))

raw_summary<-data.frame(File=c("twitter.txt", "blogs.txt", "news-
editted.txt"),
                        Lines=c(length(data_twit), length(data_blog),
length(data_news)),
                        Words=c(wcT, wcB, wcN),
                        WordsLine=c(round(wcT/length(data_twit)),
round(wcB/length(data_blog)),
                                              round(wcN/length(data_news))))
kable(raw_summary, caption="Summary of Raw Data Files.")
```

Summary of Raw Data Files.

File	Lines	Words	WordsLine
twitter.txt	2360148	28013398	12
blogs.txt	899288	36434844	41
news-editted.txt	1010242	33362288	33

The input raw datafiles consist of vectors of character vectors(lines) from each source. One thing evident at this level is that there is a considerable difference in the average number of words per line(WordsLine), especially for the Twitter data source. This isn't surprising that they are shorter due to the character length limits of Tweets. Since I will most likely have to sample data, due to computational constraints, and that it will be easiest to sample lines from the original datasets, I will need to keep this words per line difference in mind in my later sampling strategies for creating training data.

Data Sampling and Cleanup

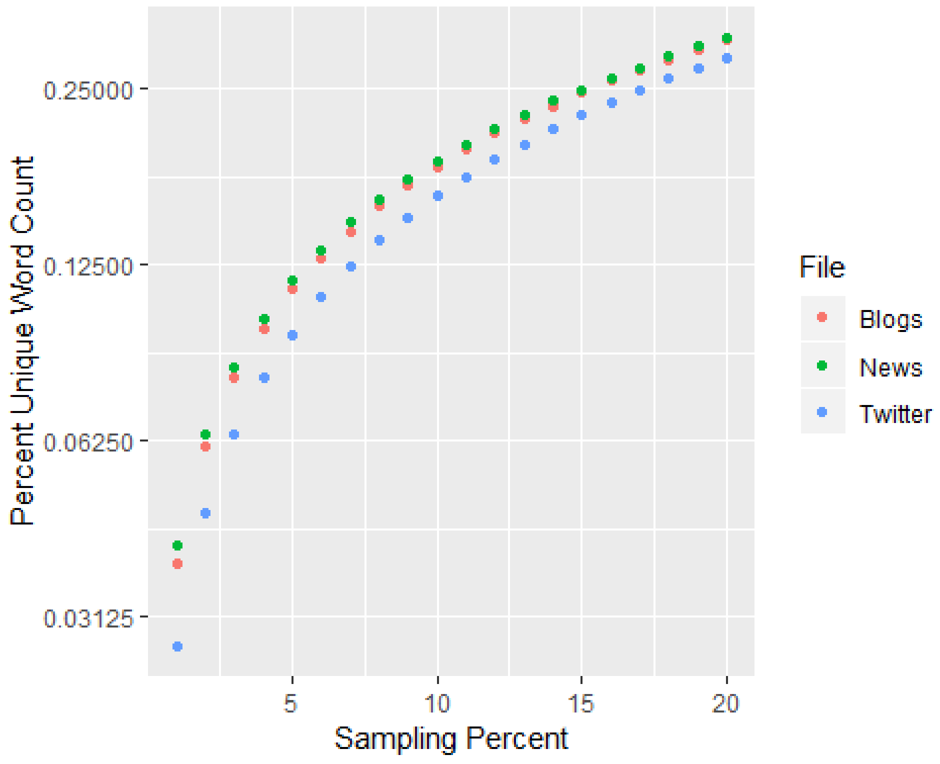
There are 2360148 lines in the twitter data, 899288 in the blog data, and 1010242 in the news data. Sampling of the data is required due to computational constraints of later analyses. It would be interesting to assess if we could use the dropoff rate of unique words to guide the amount of data to sample. To do this, I sampled between 1-20% of the lines from each of the raw data files and caculated the percent of unique words in the sample.

```
uniqWords<-function(vec, range){
  # Pretty sure the range below is wrong but it works for these cases.
  uniq<-vector(length=ceiling(((range[2]-range[1])+1)/range[3]))
  count=0
  for(i in seq(range[1], range[2], by=range[3])){
    count=count+1
    sample<-sample(vec, round((i/100)*length(vec)))
    text<-paste(sample, collapse='')
    text<-preprocess(text, remove.punct = T, remove.numbers = T)
    uniq[count]<-length(unique(unlist(strsplit(text, " "))))
  }
  return(uniq)
}
# sample data files from 1-20%
tUniq<-uniqWords(data_twit, c(1, 20, 1))
bUniq<-uniqWords(data_blog, c(1, 20, 1))
nUniq<-uniqWords(data_news, c(1, 20, 1))

tMax<-uniqWords(data_twit, c(100, 100, 1))
```

```
bMax<-uniqWords(data_blog, c(100, 100, 1))
nMax<-uniqWords(data_news, c(100, 100, 1))

df<-data.frame(File=c(rep.int(c("Twitter"), 20), rep.int(c("Blogs"), 20),
rep.int(c("News"), 20)),
               Percent=rep.int(1:20, 3), Unique=c(tUniq, bUniq, nUniq),
               PerUniq=c((tUniq/tMax), (bUniq/bMax), (nUniq/nMax)))
ggplot(df, aes(x=Percent, y=PerUniq, color=File)) + geom_point() +
scale_y_continuous(trans='log2')+
  xlab("Sampling Percent") + ylab("Percent Unique Word Count")
```



There are a couple interesting points that can be taken from this plot. One, the Twitter datafile yields slightly less unique words that the others for the same sampling rate. i.e. Smaller vocabulary used. This indicates that it is reasonable to tailor the sampling rate for this data to match it’s contribution to the other two data sources. Two, sampling at 5% of the data lines yields only about 12% of the unique words indicating that there could be significant gains to be made at higher sampling rates. I will keep this in mind for future analysis but, due to the limits of my computational resources, I will sample 3% of the Blogs & News data and 4% of the Twitter data. I will also standardize the letter case and strip out punctuation and numbers from the text using the preprocess() function from the ngram package.

```
dataT<-sample(data_twit, round(0.04*length(data_twit)))
dataB<-sample(data_blog, round(0.03*length(data_blog)))
dataN<-sample(data_news, round(0.03*length(data_news)))
# Remove files and garbage collect
```

```r
rm(data_twit, data_blog, data_news)
foo<-gc()

# Standardize letter case and strip out punctuation and numbers
dataT<-preprocess(paste(dataT, collapse=''), remove.punct = T, remove.numbers
= T)
dataB<-preprocess(paste(dataB, collapse=''), remove.punct = T, remove.numbers
= T)
dataN<-preprocess(paste(dataN, collapse=''), remove.punct = T, remove.numbers
= T)
```

Exploratory Analysis

The frequency of n-grams from each of these sources will be generated, compared, and contrasted.

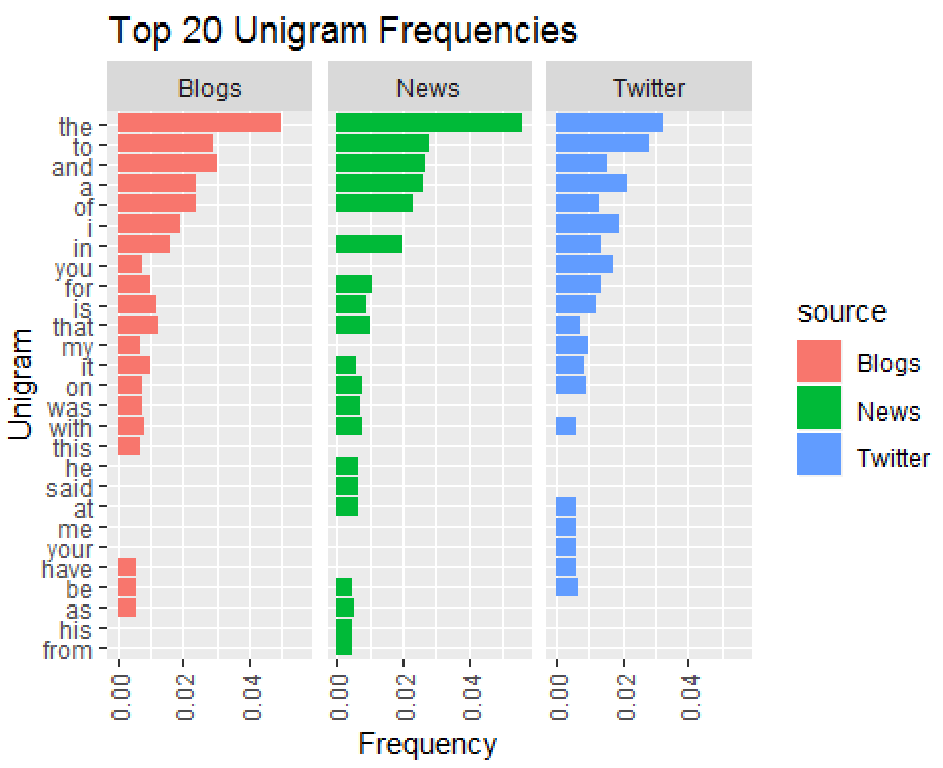
```r
ng1T<-summary(factor(unlist(strsplit(dataT, " "))), maxsum=1000)
ng2T<-ngram(dataT, n=2)
ng3T<-ngram(dataT, n=3)
ng4T<-ngram(dataT, n=4)

ng1B<-summary(factor(unlist(strsplit(dataB, " "))),maxsum=1000)
ng2B<-ngram(dataB, n=2)
ng3B<-ngram(dataB, n=3)
ng4B<-ngram(dataB, n=4)

ng1N<-summary(factor(unlist(strsplit(dataN, " "))),maxsum=1000)
ng2N<-ngram(dataN, n=2)
ng3N<-ngram(dataN, n=3)
ng4N<-ngram(dataN, n=4)

# Generate uni-gram frequency dataframe
# Convert to freqs
ng1T<-ng1T/sum(ng1T)
ng1B<-ng1B/sum(ng1B)
ng1N<-ng1N/sum(ng1N)

# Generate unigram dataframe for plotting
df1<-data.frame(source=c(rep.int("Twitter", 20), rep.int("Blogs", 20),
rep.int("News", 20)),
               ngram=c(names(ng1T)[1:20], names(ng1B)[1:20],
names(ng1N)[1:20]),
               freq=c(ng1T[1:20], ng1B[1:20], ng1N[1:20]))
# Plot
ggplot(df1, aes(y=freq, x=reorder(ngram, freq),
fill=source))+geom_bar(stat="identity")+coord_flip()+
  facet_wrap(~source) + ggtitle("Top 20 Unigram
Frequencies")+xlab("Unigram")+ylab("Frequency")+
  theme(axis.text.x=element_text(angle=90, hjust=1, vjust=0.5))
```

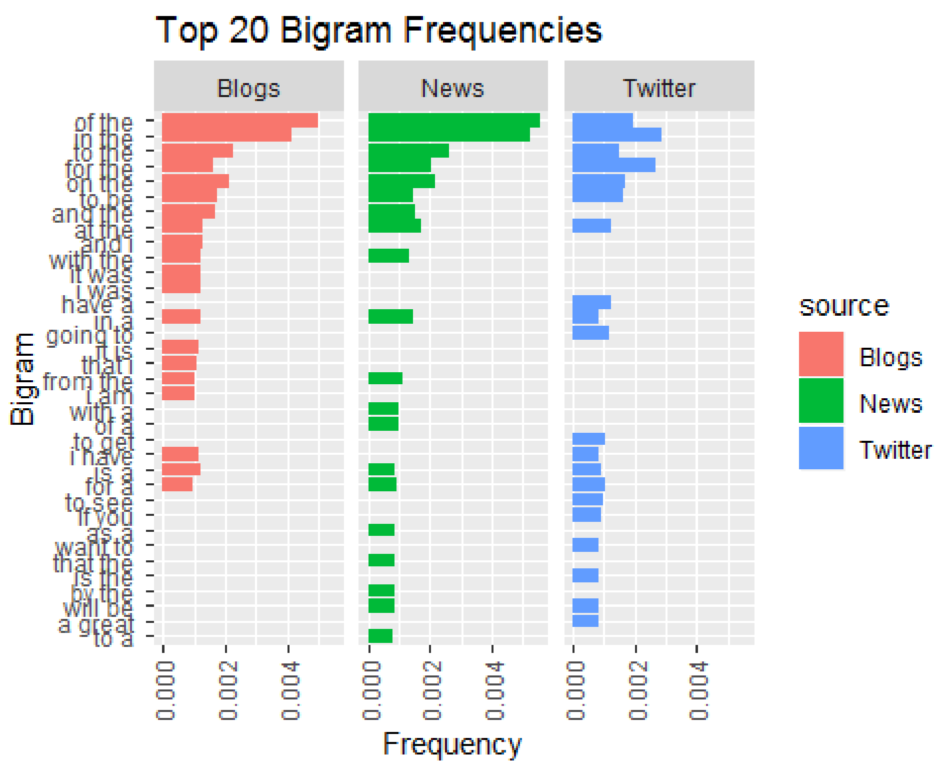


Some interesting differences between the data sources are detectable at the unigram level. For example, "I, you, my" are not in the list of top 20 common words for News but are for Twitter and Blogs. This is possibly due to the avoidance of first person writing in news articles.

```
# Generate bigram dataframe
p2T<-get.phrasetable(ng2T)
p2B<-get.phrasetable(ng2B)
p2N<-get.phrasetable(ng2N)

df2<-data.frame(source=c(rep.int("Twitter", 20), rep.int("Blogs", 20),
rep.int("News", 20)),
               ngram=c(p2T[1:20,c("ngrams")],
                       p2B[1:20,c("ngrams")],
                       p2N[1:20,c("ngrams")]),
               freq=c(p2T[1:20,c("prop")],
                       p2B[1:20,c("prop")],
                       p2N[1:20,c("prop")]))

# Plot
ggplot(df2, aes(y=freq, x=reorder(ngram, freq),
fill=source))+geom_bar(stat="identity")+coord_flip()+
  facet_wrap(~source) + ggtitle("Top 20 Bigram
Frequencies")+xlab("Bigram")+ylab("Frequency")+
  theme(axis.text.x=element_text(angle=90, hjust=1, vjust=0.5))
```

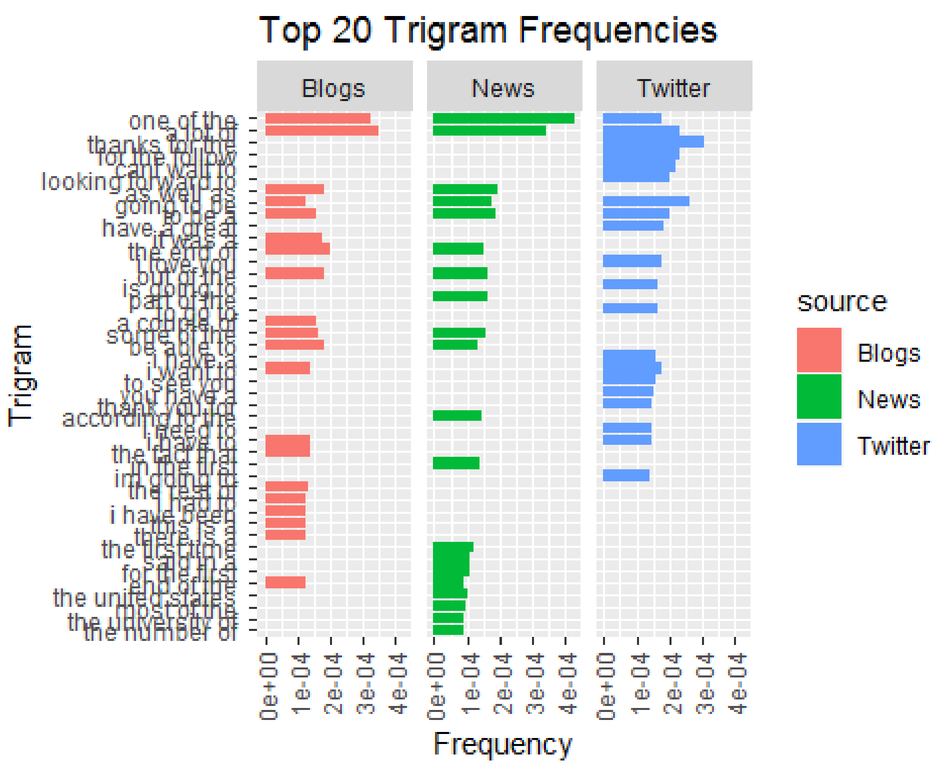


All 3 sources share their top 6 bigrams but the previous pattern of the pronouns “I”" and “you”" being common in Twitter and Blogs, as opposed to the News text, seems to break down. Generally, all three data sources appear to have distinct patterns of common bigrams. It is interesting to note that the most common 1-2 bigrams in Blogs and News have much higher frequencies than the relatively flat frequencies in the Twitter datasource.

```
# Generate trigram dataframe
p3T<-get.phrasetable(ng3T)
p3B<-get.phrasetable(ng3B)
p3N<-get.phrasetable(ng3N)

df3<-data.frame(source=c(rep.int("Twitter", 20), rep.int("Blogs", 20),
rep.int("News", 20)),
               ngram=c(p3T[1:20,c("ngrams")],
                       p3B[1:20,c("ngrams")],
                       p3N[1:20,c("ngrams")]),
               freq=c(p3T[1:20,c("prop")],
                       p3B[1:20,c("prop")],
                       p3N[1:20,c("prop")]))

# Plot
ggplot(df3, aes(y=freq, x=reorder(ngram, freq),
fill=source))+geom_bar(stat="identity")+coord_flip()+
  facet_wrap(~source) + ggtitle("Top 20 Trigram Frequencies")+xlab("Trigram")
+ ylab("Frequency")+
  theme(axis.text.x=element_text(angle=90, hjust=1, vjust=0.5))
```



At the trigram level, some common short English phrases can be encapsulated and the most common ones do seem to fit the different data sources. For example, personal messages like “I love you”, “thanks for the”, “cant wait to”, and “looking forward to” are very common in the Twitter dataset but are not present in the 20 most common trigrams for News and Blogs. Likewise, “the united states” is in the list from the News datasource but not the others.

Summary and Algorithm Plan

While it might be easy to assume all three datasources are comparable, they utilize distinct subsets of the English language to different amounts. For example, the Twitter data consists of shorter lines, more redundant vocabulary, and a greater usage of some personal pronouns. My current algorithm/app plan is to calculate n-gram frequencies from this data and store it in a structure that will allow quick lookups. As the user types in their words, the algorithm will identify the most probable next word based on the preceding words. For example, the user types in “I love”, the algorithm finds the most likely trigram starting with “I love” and provides the next word. Therefore, it will be important to keep in mind that these data sources have different n-gram frequencies and would produce models with different underlying word probabilities.