Data Science Capstone Milestone Report

Finbark

17 March 2019

Introduction

This milestone report shows the results of exploritory analysis done on the initial data set. It then moves on to discuss the forthcomming predictve model.

Getting the data

This section outlines how the data was obtained and sampled.

Load the initial data

The data is downloaded and unzipped to the local directory. But only if it hasn't already been downloaded

```r
if (!file.exists("./data.zip")) {
    URL <- "http://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-SwiftKey.zip"
    download.file(URL, destfile = "data.zip", method = "curl")
    unzip("data.zip")
}
```

Read and sample data

To make things faster, the data is loaded but only a sample is taken

```r
start <- function() {
    files <- list.files(path = './final/en_US/', pattern = "*.txt")
    files <- lapply(files, function(x) paste('./final/en_US/', x, sep = ""))
    dataSample <- unlist(lapply(files, function(x) loadAndSampleData(x, 20000)))
}

loadAndSampleData <- function(data, sample_size) {
    sampleData <- readLines(data, warn = FALSE, encoding = "UTF-8", skipNul = TRUE)
    sampleData <- selectiveSample(sampleData, sample_size)
}

## Used to handle when the desired sample size is greater than the data set.
selectiveSample <- function(data, sample_size) {
    if (length(data) < sample_size) {
        data
    }
    else {
        sample(data, sample_size)
    }
}
```

```r
data_sample <- start()
```

Initial exploration

This section shows the initial exploration on the the three main data sets and then the sampled data set.

Blog data set

```r
data.frame("Source" = "Sample data",
           "Total lines" = length(blogs),
           "Total words" = sum(stri_stats_latex(blogs)[4]),
           "Total characters" = sum(nchar(blogs)))
```

```
##        Source Total.lines Total.words Total.characters
## 1 Sample data      899288    37865888        208361438
```

News data set

```r
data.frame("Source" = "Sample data",
           "Total lines" = length(news),
           "Total words" = sum(stri_stats_latex(news)[4]),
           "Total characters" = sum(nchar(news)))
```

```
##        Source Total.lines Total.words Total.characters
## 1 Sample data       77259     2665742         15683765
```

Twitter data set

```r
data.frame("Source" = "Sample data",
           "Total lines" = length(twitter),
           "Total words" = sum(stri_stats_latex(twitter)[4]),
           "Total characters" = sum(nchar(twitter)))
```

```
##        Source Total.lines Total.words Total.characters
## 1 Sample data     2360148    30578891        162384825
```

Sample data set

```r
data.frame("Source" = "Sample data",
           "Total lines" = length(data_sample),
           "Total words" = sum(stri_stats_latex(data_sample)[4]),
           "Total characters" = sum(nchar(data_sample)))
```

```
##        Source Total.lines Total.words Total.characters
## 1 Sample data       30000      883714          4975188
```

Creating a clean corpus of text

The tm package is now used on the sample data to convert it into an appropriate formate for ngrams. The data is also cleaned.To avoid repetition, the results are saved so they can be loaded again later.

```r
start <- function() {
    profanity <- readLines("./profanityFilter/profanityFilter.txt")
    tokenizedData <- createTokenizedData(data_sample)
    cleanTokenizedData <- cleanTokenizedData(tokenizedData, profanity)
    documentMatrix <- TermDocumentMatrix(cleanTokenizedData)
    cleanTokenizedData
}

createTokenizedData <- function(data) {
    Corpus(VectorSource(data))
}

cleanTokenizedData <- function(data, filter) {
    removeURL <- function(x) gsub("http[[:alnum:]]*", "", x)
    cleanData <- tm_map(data, content_transformer(removeURL))
    cleanData <- tm_map(cleanData, content_transformer(tolower))
    cleanData <- tm_map(cleanData, content_transformer(removePunctuation))
    cleanData <- tm_map(cleanData, content_transformer(removeNumbers))
    cleanData <- tm_map(cleanData, stripWhitespace)
    cleanData <- tm_map(cleanData, removeWords, stopwords("english"))
    cleanData <- tm_map(cleanData, removeWords, filter)
}

saveCorpusandMatrix <- function(corpus, matrix) {
    saveRDS(corpus, file = "./Corpus.RData")
}

corpus <- start()
```

Creating Ngrams

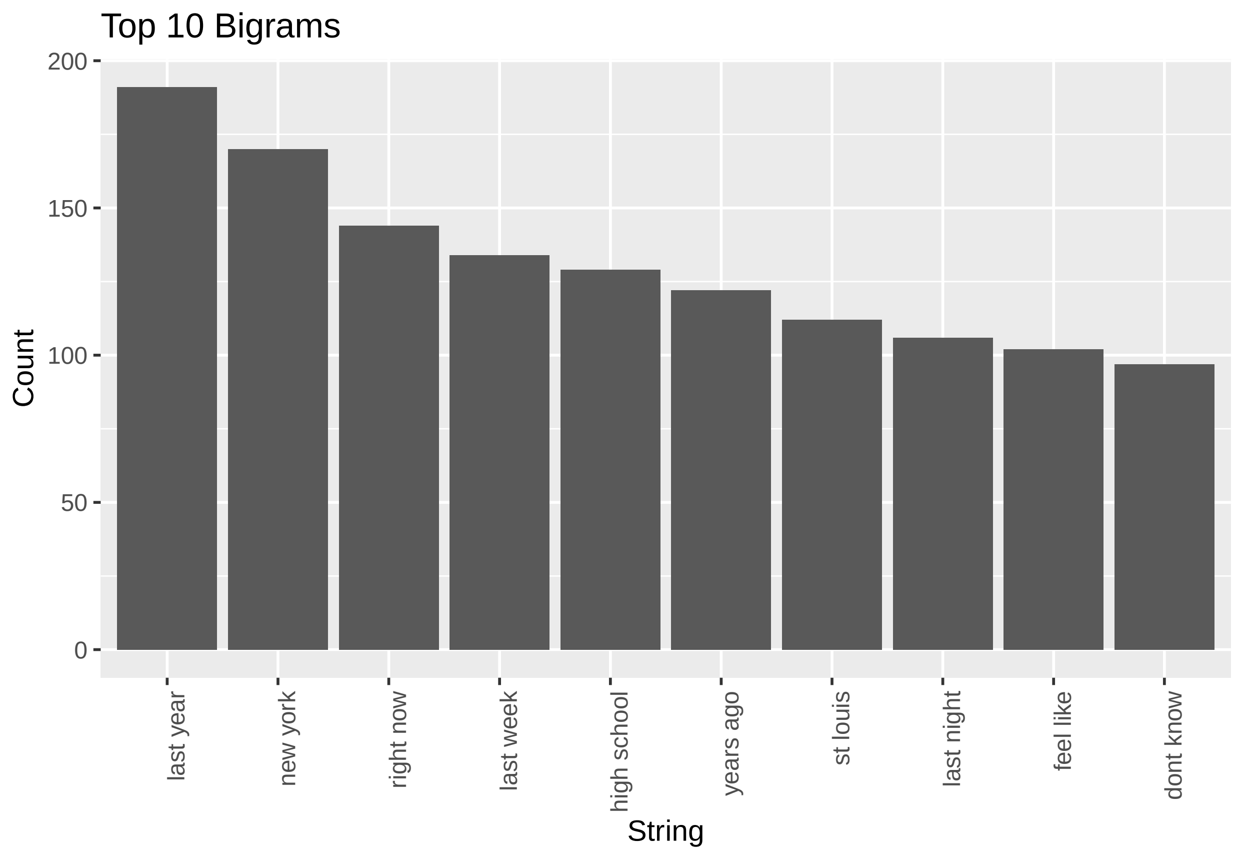
Before we do the last bit of analysis, we need to create sets of ngrams from the corpus. They are saved for ease of access later.

```r
start <- function() {
    bigram <- ngramTokenizer(corpus, 2)
    saveRDS(bigram, file = "./ProcessData/bigram.RDS")
    rm(bigram)
    trigram <- ngramTokenizer(corpus, 3)
    saveRDS(trigram, file = "./ProcessData/trigram.RDS")
    rm(trigram)
    quadgran <- ngramTokenizer(corpus, 4)
    saveRDS(quadgran, file = "./ProcessData/quadgram.RDS")
}

ngramTokenizer <- function(corpus, ngramCount) {
    ngram <- NGramTokenizer(corpus,
                            Weka_control(min = ngramCount, max = ngramCount, delimiters = " \\r\\n\\t.,
    ngram <- data.frame(table(ngram))
    ngram <- ngram[order(ngram$Freq, decreasing = TRUE),][1:10,]
    colnames(ngram) <- c("String","Count")
    ngram

}
```

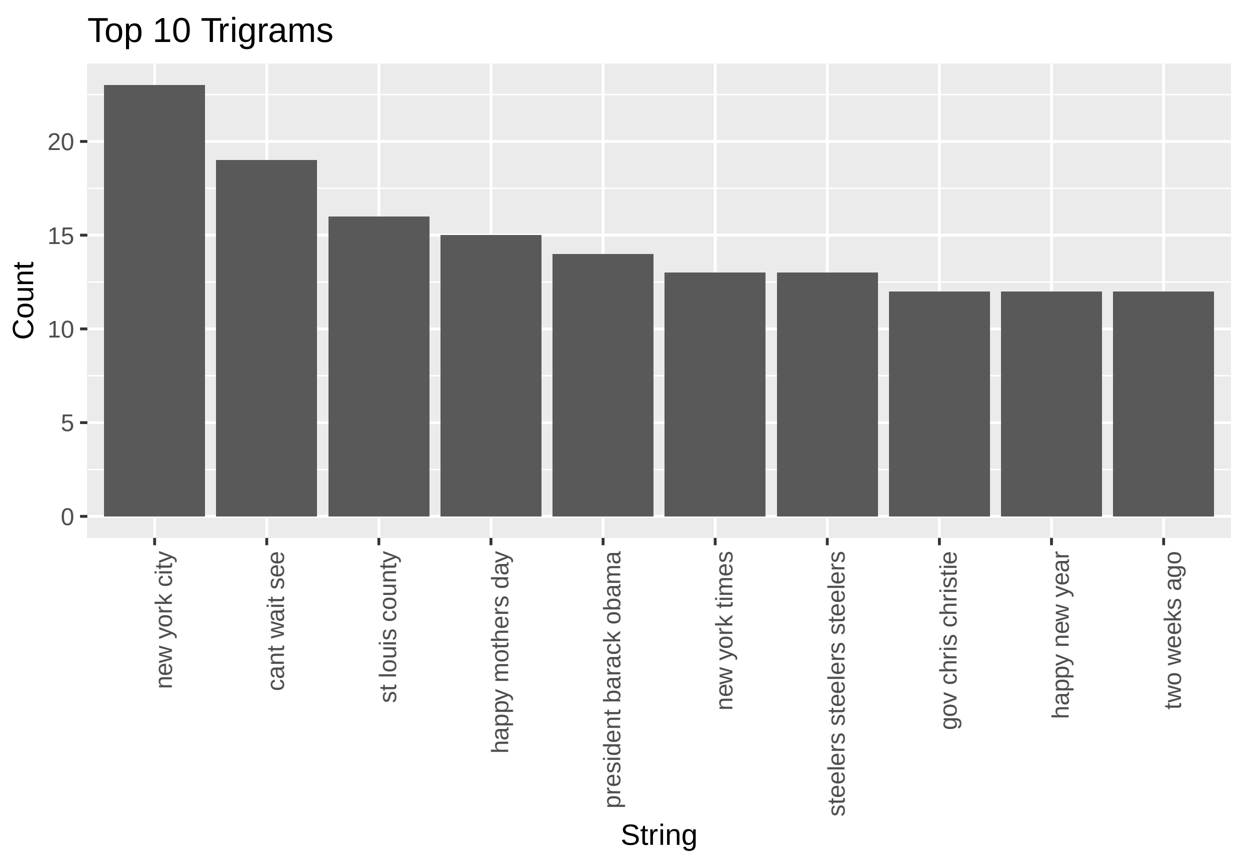
Ngram exploration

We now move to explore the three types of Ngrams, and show the 10 most frequent terms in each file.

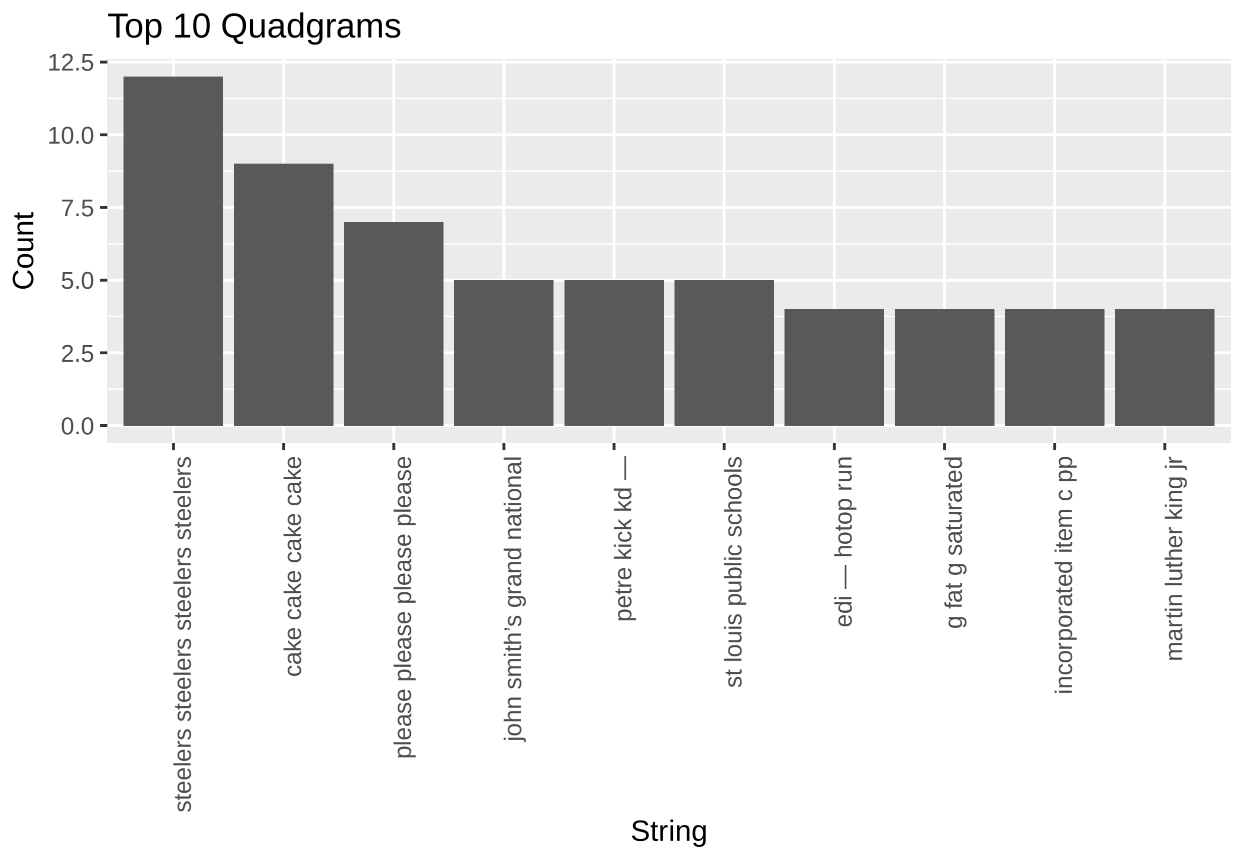
```
bigram_top10 <- transform(bigram[1:10,], String = reorder(String, order(Count, decreasing = TRUE)))
ggplot(data = bigram_top10, aes(x = String, y = Count)) +
    geom_col() +
    theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
    ggtitle("Top 10 Bigrams")
```



```
trigram_top10 <- transform(trigram[1:10,], String = reorder(String, order(Count, decreasing = TRUE)))
ggplot(data = trigram_top10, aes(x = String, y = Count)) +
    geom_col() +
    theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
    ggtitle("Top 10 Trigrams")
```



```
quadgram_top10 <- transform(quadgram[1:10,], String = reorder(String, order(Count, decreasing = TRUE)))
ggplot(data = quadgram_top10, aes(x = String, y = Count)) +
    geom_col() +
    theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
    ggtitle("Top 10 Quadgrams")
```



Findings

From the initial exploration, we can see that there are some comminalities between the three sets of ngrams; for example, 'new york' and 'new york city'. But for the most part, the three sets are different.

Second, and more generally, the whole data set is very large, and it is impractical to build ngrams from the whole data set. A sub-set with a total size of 60,000 lines was used for this exercise. But given more time, a bigger sub-set would be advisiable, to help with a better prediction at the end.

Next steps

The main next step is to develop a prediction algrorithm from the data ngram data sets.

One essential step to doing this will be to restructure the data so instead of a full string in a column, each word in the ngram is its own column. This will be achieved using the str_split_fixed function from the stringr package.