Next word prediction: A text mining and NLP Project

JHU Data science on Coursera

Rahul Nayak

Motivation

The goal of this project is for the student to learn applied text mining and natural language processing in R and create a practical next word predictor in the form of a shiny app,taking a meaningful sequence of words as an input, could predict the word most likely to follow or complete such a sequence. This functionality could be used to speed user typing by suggesting the next word to the user and let them select it rather than having to type it themselves. Another potential application is search query autocomplete.

Data

The data is provided in the form of a collection of tweets, a collection of blogs and a collection of news arcticles. The data is accessible from here (https://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-SwiftKey.zip).

The data is availabe for a few different languages, but we will be using the englis language data for now. The english data comes in three files:

1. **en_US.blogs.txt**

- Size: 200 MB
- Lines: 899288
- Wordcount: 37334131

Line summary

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     1.0    47.0   157.0   231.7   331.0 40835.0
```

2. **en_US.news.txt**

- Size: 196 MB
- Lines: 2643969
- Wordcount: 2643969

Line summary

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       2     111     186     203     270    5760
```

3. **en_US.twitter.txt**

- Size: 159 MB
- Lines: 2360148
- Wordcount: 30373543

Line summary

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     2.0    37.0    64.0    68.8   100.0   213.0
```

It is known that twitter has a 140 character limit, however from the summary it is evident that there is atleast one tweet that breaks this rule.Here is that tweet:

```
## [1] "It's time for you to give me a little bit of lovin'ï¼\210ã\201•ã\201\201ã\20
1¡ã,‡ã\201£ã\201¨ã\201¯ã\201,ã\201ªã\201Ÿã\201®æ„›ã,’ã\201¡ã,‡ã\201†ã\201 ã\201„ï¼‰Ba
by, hold me tight and do what I tell youï¼\201ï¼\210ãƒ\231ã,¤ãƒ“ãƒ¼æŠ±ã\201\215ã\201—
ã,\201ã\201¦ç§\201ã\201Œè¨\200ã\201†ã,\210ã\201†ã\201«ï¼\201ï¼‰"
```

Hence we have to undertake a massive data cleaning campaign!

Apart from this additional data in the form of tweets, blogs, product reviews and news headlines were also analyzed.

Data Cleaning

To clean up the text data, we performed the following (ordered) list of operations using the tm package (https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwinvOLa8_bpAhWCCuwKHf7TDRkQFjAZegQIARAB&url=https%3A%2F%2Fcran.r-project.org%2Fweb%2Fpackages%2Ftm%2Ftm.pdf&usg=AOvVaw1QBcgq3DCKNvLrLslqAu4z).

1. Remove all symbols not alphanumerics.
2. Convert to lower case.
3. Remove numbers.
4. Remove extra white space.
5. remove profanity laced sentences.

A list of words and phrases that are considered profanity was obtained from here (https://raw.githubusercontent.com/shutterstock/List-of-Dirty-Naughty-Obscene-and-Otherwise-Bad-Words/master/en).It was decided to remove the entire line that contains the profanity. Words that were previously separated by one or more profane words could then become adjacent. Hence, down the line, we would also end up generating N-grams that simply weren’t there in the original data.

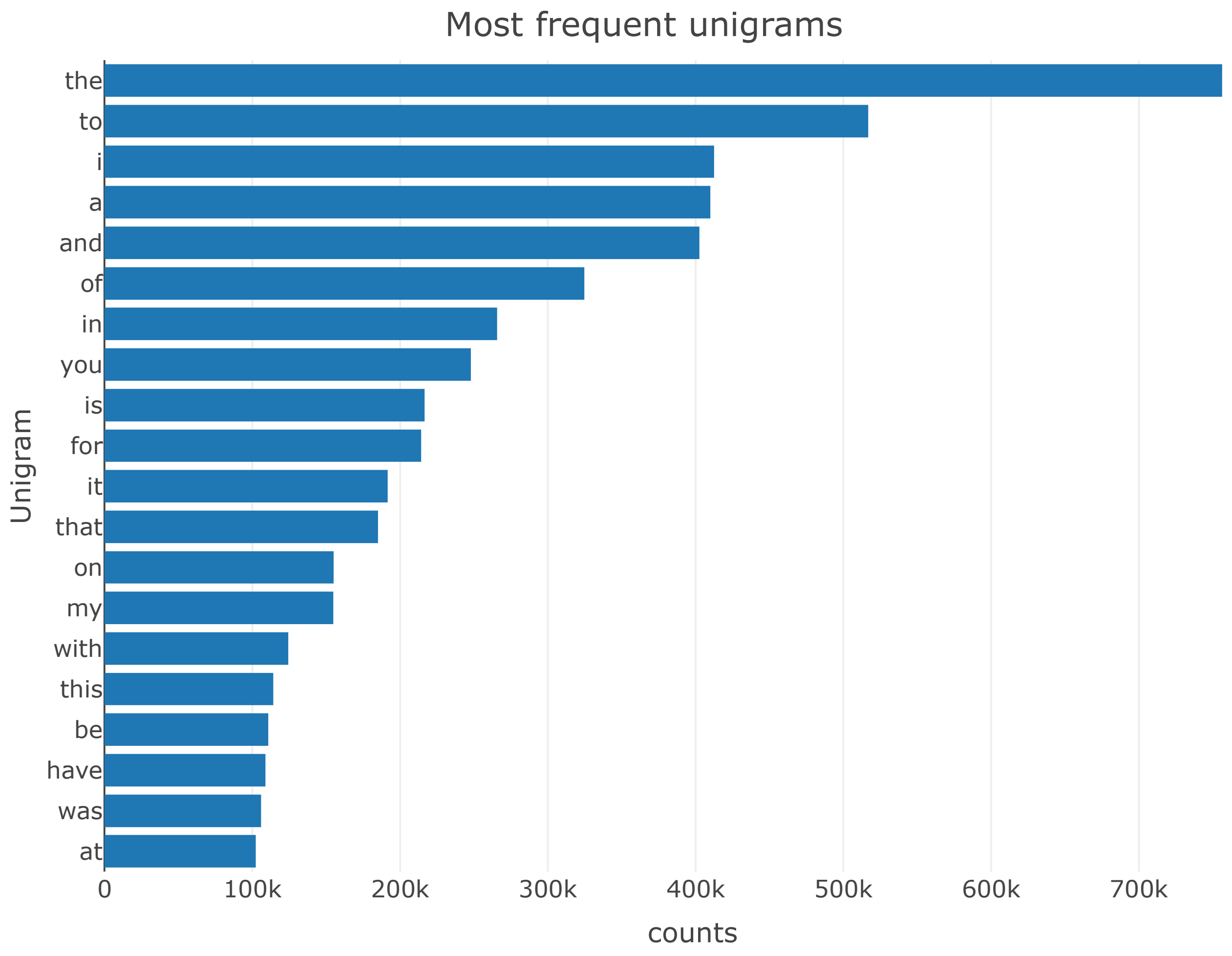
Note that we did not remove stopwords (most common words of the English language). Our goal being to create a next-word predictor based on the words typed by the user, removing stopwords would result in the creation of a large number of meaningless N-grams.

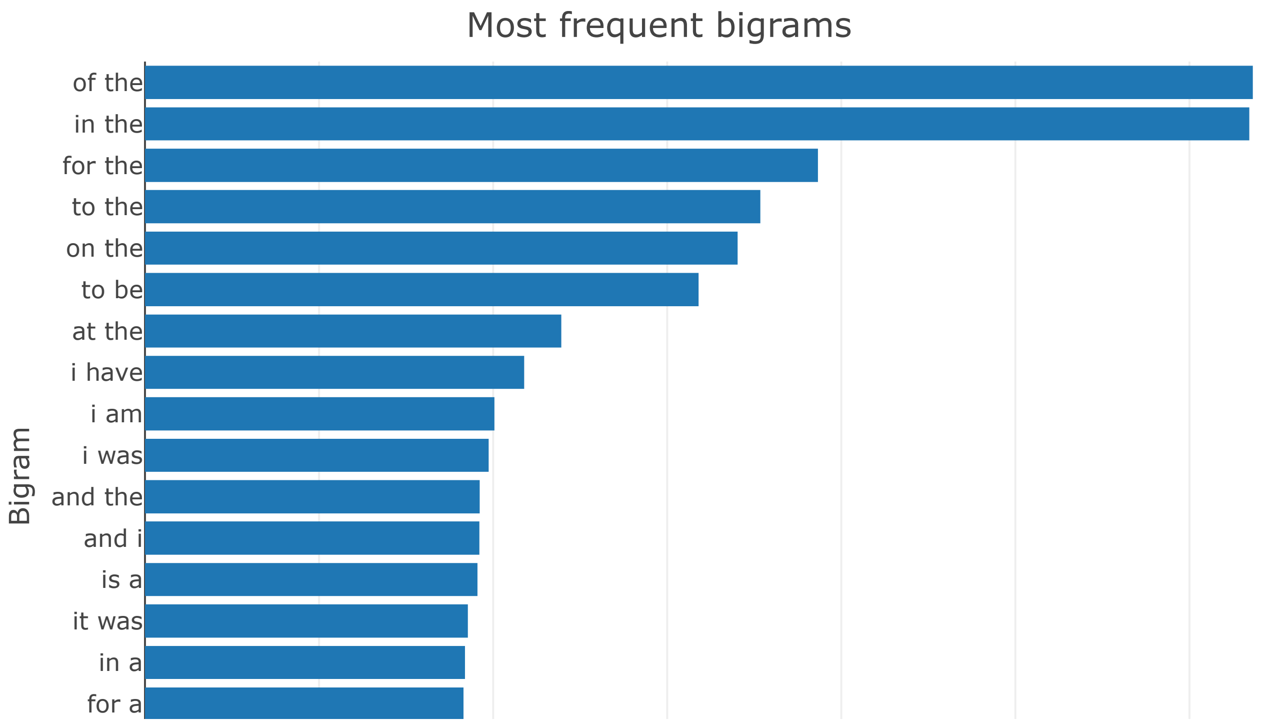
Bulding N-Gram frequency tables

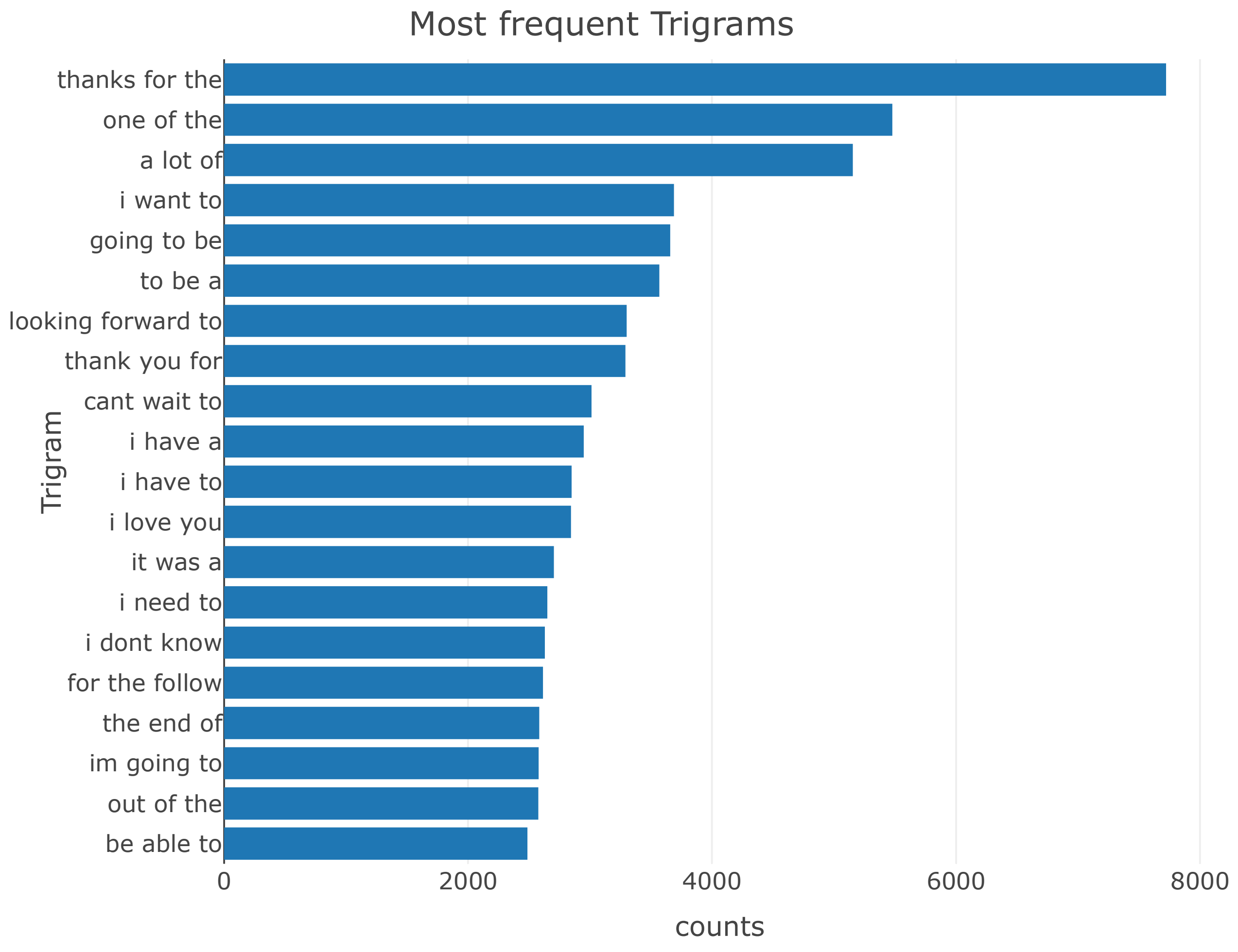
After data cleaning, we built term-frequency matrices for 5-grams, all the way down to unigrams, using the function textcnt() from the tau (https://cran.r-project.org/web/packages/tau/index.html) R package.

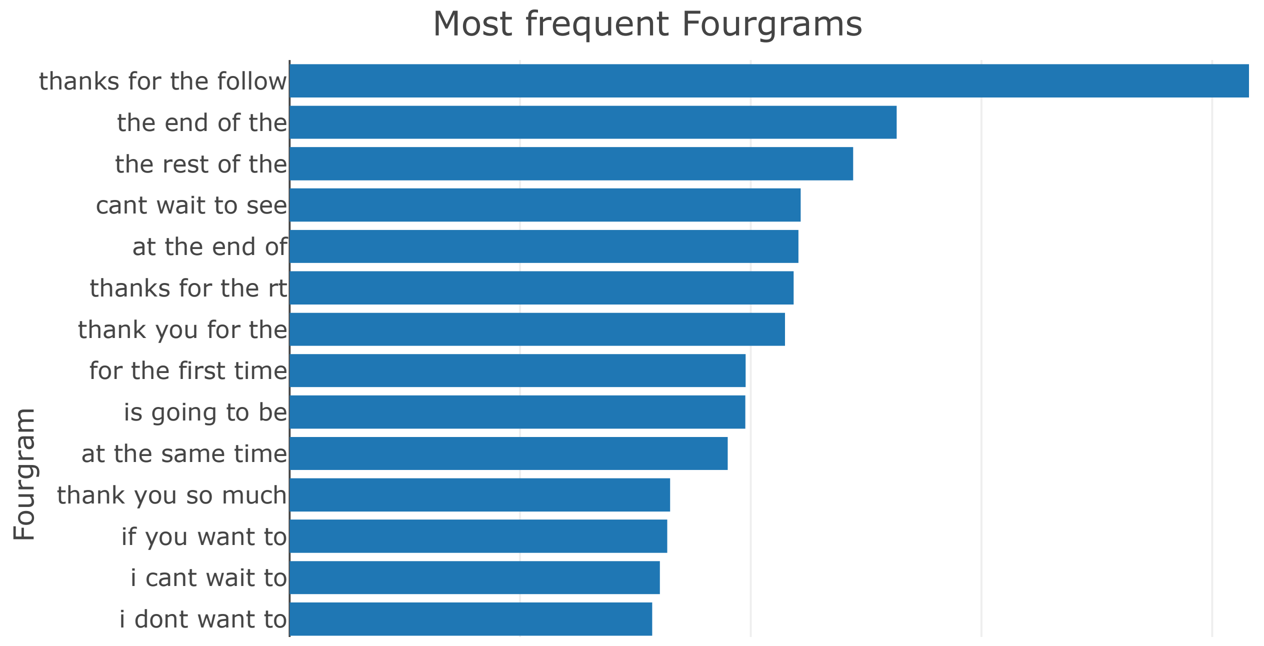
Most Frequent N-Grams

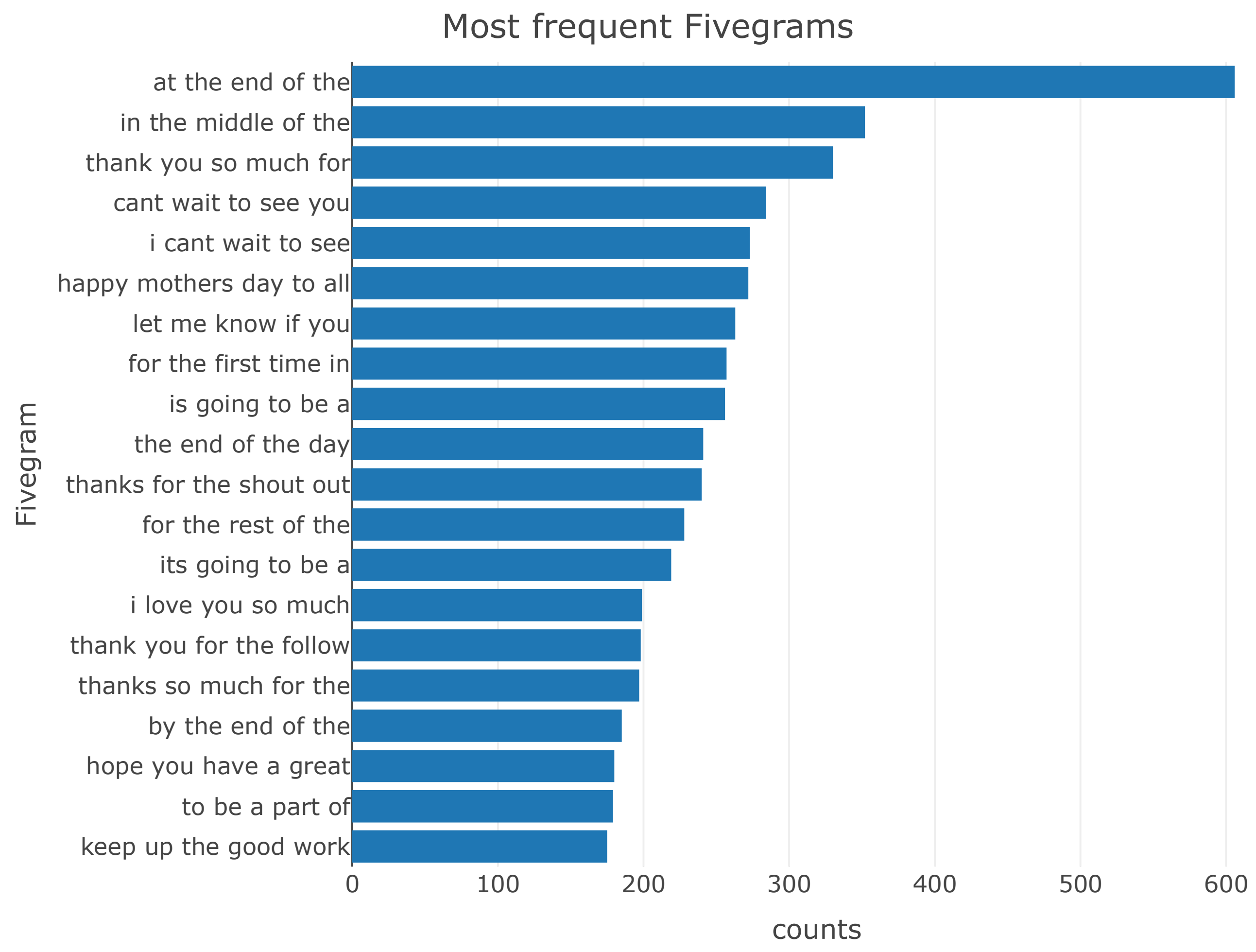
The following plots show the most common N-grams mined from our data: (At the time of writing this report, it was decided to split the total data into 1 million entities from the available 3 million+ entities with the remaining data processed later.)






Most frequent Trigrams
thanks for the
one of the
a lot of
i want to
going to be
to be a
looking forward to
thank you for
cant wait to
i have a
i have to
i love you
it was a
i need to
i dont know
for the follow
the end of
im going to
out of the
be able to
Trigram
0
2000
4000
6000
8000
counts


Most frequent Fourgrams
thanks for the follow
the end of the
the rest of the
cant wait to see
at the end of
thanks for the rt
thank you for the
for the first time
is going to be
at the same time
thank you so much
if you want to
i cant wait to
i dont want to
Fourgram



N-gram cleaning:

N-gram frequency tables contain few terms with high frequency counts and many terms with low frequency counts. Ultra-low frequency items use a significant amount of memory while adding little value to our language model (their probability to occur being so low, our predictor is very unlikely to show them as predictions to the user). Hence, we can save a significant amount of memory if we only store terms with some minimum count. Here, we set this threshold at 4 – any N-gram with a count less than four was pruned.

Next steps

There is a need to implement next word prediction using backoff algorithms (http://en.wikipedia.org /wiki/Katz%27s_back-off_model). For this we will use the Stupid Backoff algorithm. (https://www.aclweb.org /anthology/D07-1090.pdf).

Using the generated ngram tables and this algorithm we will implement next word prediction. The successfull implementation will be used to build the final product a shiny app. We want fast prediction and accurate prediction and hence it is needed to optimise the size of the ngram models. Also the scores for stupid backoff models can be precalculated to save computation time.

Prediction algorithm

There is a need to implement next word prediction using backoff algorithms (http://en.wikipedia.org/wiki/Katz%27s_back-off_model). For this we will use the Stupid Backoff algorithm (https://www.aclweb.org/anthology/D07-1090.pdf).

Using the generated ngram tables and this algorithm we will implement next word prediction.To rank next-word candidates, we use a mechanism first described as Stupid Backoff. In their seminal paper (http://www.aclweb.org/anthology/D07-1090.pdf), the authors define the following scoring function:

$$S(w_i|w_{i-k+1}^{i-1}) = \begin{cases} \frac{\text{count}(w_{i-k+1}^{i})}{\text{count}(w_{i-k+1}^{i-1})} & \text{if count}(w_{i-k+1}^{i}) > 0 \\ \lambda S(w_i|w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

The stupid backoff score function

Our use of this scoring approach can be summarized as follows:

```
if (candidateIs5gram) {
score = matched5gramCount / input4gramCount
} else if (candidateIs4gram) {
score = 0.4 * matched4gramCount / input3gramCount
} else if (candidateIs3gram) {
score = 0.4 * 0.4 * matched3gramCount / input2gramCount
} else if (candidateIs2gram) {
score = 0.4 * 0.4 * 0.4 * matched2gramcount / input1gramCount
}
```

Based on the obtained scored the top three words with highest scores are presented as next word prediction.