Technical Report: A Shiny Predictive Text Product

Aliakbar Safilian[*]

June 09, 2019

Contents

1 Introduction

The *Shiny Predictive Text App*[1] (Fig. 1) is a web-based application suggesting words the end user may wish to insert in a text field. The current report describes the technical aspect of the product.

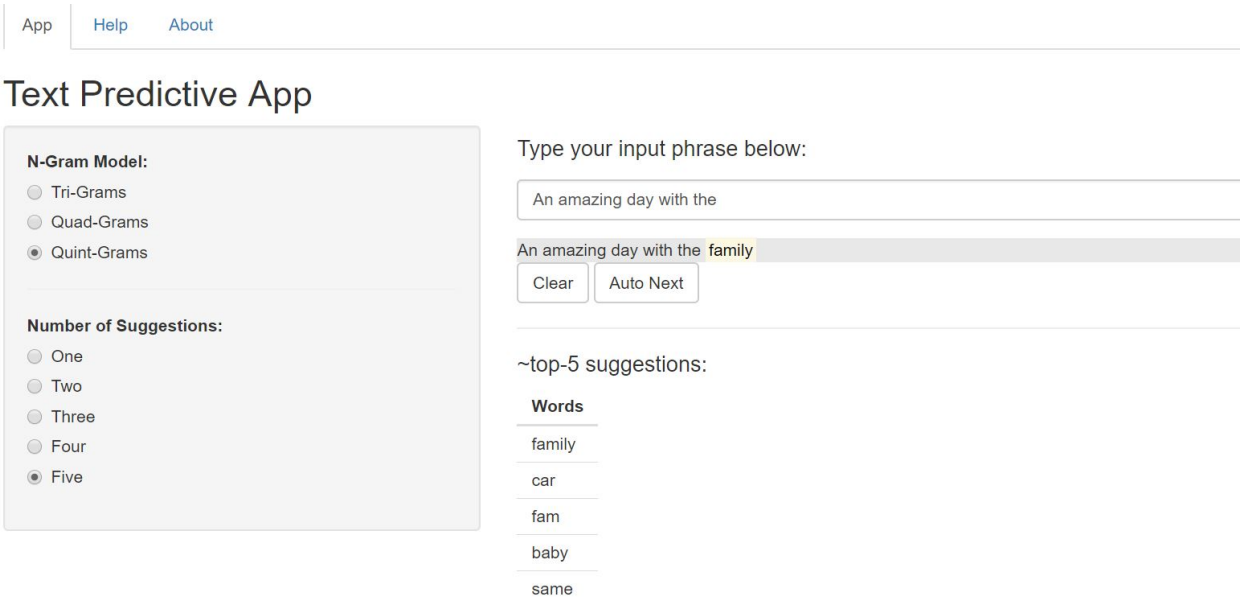

Figure 1: The Text Predictive Shiny App

[*]a.a.safilian@gmail.com

[1]https://asafilian.shinyapps.io/txt_predict_app

We analyze a large corpus of text documents (more than *4 million lines*, and over *102 million words* – See Table. 1) to discover the relationship between words. The process involves cleaning and analyzing text data, then building and sampling from a predictive text model. Finally, we will build our shiny app.

The basic training data for this project has been provided by Swiftkey[2]. The data[3] is from a corpus called HC Corpora. There are four different databases, each for one specific language (German, English, Finnish, and Russian). In this project, we deal only with the English database. There are the following textual files in the English database:

- `en_us.blogs.txt`
- `en_us.news.txt`
- `en_us.twitter.txt`

One of the main challenges we have in this project is the *limited computational resources* (memory and time). We have to make a tradeoff in between size and runtime. For example, an algorithm that requires a lot of memory may run faster, while a slower algorithm may require less memory. Therefore, we need to find the right balance between the two in order to provide a good experience to the user. We have tried nine different models with two variable factors:

- A *fraction* of the original corpus (50%, 60%, or 70%)
- An *n-gram model* (*tri-gram*, *quad-gram*, or *quint-gram*)[4]

Please see Sect. 4, where we describe why we decided to work on a 60% fraction of the original corpus and implemented a quint-gram model on it.

After getting a random sample of the corpus, we perform the following *preprossing steps* on the sample data:

- *lower-case* conversion
- removing *hyphens*
- removing twitter and other *symbols*
- removing *separators* (white-spaces)
- removing *punctuations*
- removing *numbers*
- removing *profanities*
- removing *non-English* words

We then extract *uni-grams* (words), *bi-grams* (two consecutive words), and *tri-grams* (three consecutive words), *quad-grams* (four consecutive words), and *quint-grams* (five consecutive words) from the clean data, and represent several interesting results about them. We perform some exploratory analysis to understand the distributions of term frequencies in n-grams.

Next, we build an *n-gram model* to predict next words given a phrase. To further optimize the memory usage, we do several more preprocessing on n-grams. We follow the *Stupid Backoff method*[5] in building our model. That is, to predict the next word, we first use the quint-gram probability. If we do not have enough of a quint-gram count to make it, we back-off and use the quad-gram probability. If there still is not enough of a quad-gram count, we use the tri-gram probability. If we fail again, the algorithm tries bi-gram and uni-grams probabilities. To calculate the n-gram probabilities, we use the *Kneser-Ney smoothing method*[6].

We evaluate the quality (*precision*, *average runtime*, and *memory consumption*) of several models by scripts and a testing data provided by Hernán Foffani[7]. Accordingly, we select a model that works the best.

The structure of the rest of the report is as follows: In Sect. 2, we tokenize and clean the data. We analyze the bi-, tri-, quad-, and quint-grams extracted from the clean data in Sect. 3. Sect. 4 describes our language

[2] https://www.microsoft.com/en-us/swiftkey/about-us

[3] The data can be downloaded at https://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-SwiftKey.zip.

[4] tri-gram = 3-gram, quad-gram = 4-gram, quint-gram = 5-gram

[5] Brants et al (2007). "Large language models in machine translation." https://www.aclweb.org/anthology/D07-1090

[6] Ney and Reinhard Kneser (1994). "On structuring probabilistic dependences in stochastic language modelling." https://www.sciencedirect.com/science/article/pii/S0885230884710011

[7] The Git repository at https://github.com/jan-san/dsci-benchmark

model. One can use the scripts in Scripts to reproduce the results.

2 Preprocessing and Uni-Grams

We first load the data, and get a general picture of the data:

Table 1: Raw Data - Information

Data	Size	Lines	Words	Range nchars	Avg nchars
Blogs	210.16 MB	899,288	37,334,131	1 - 40,833	229.99
Twitter	167.11 MB	2,360,148	30,373,583	2 - 213	68.8
News	205.81 MB	1,010,242	34,372,530	1 - 11,384	201.16
Corpus	583.08 MB	4,269,678	102,080,244	1 - 40,833	499.95

Note:
Words: approximate in million
Range nchars: range of number of chars in lines
Avg nchars: avgerage number of chars in lines

As we see in Table. 1, the original corpus includes over 40 million lines and over 102 million words. The main constraint in this project is the computational resource (i.e., time and memory). To alleviate this issue, we get a random sample fraction. Table. 2 represents a random sample fraction of 60% from the data. Since the average numbers of characters have not changed much, the sample looks reasonable.

Table 2: Random Sampled Raw Data - Information

Data	Lines	Words	Range nchars	Avg nchars
Blogs	539,572	22,396,051	1 - 40833	230.02
Twitter	1,416,088	18,223,509	2 - 211	68.8
News	606,145	20,629,016	1 - 8949	201.21
Corpus	2,561,805	87,142,164	1 - 40833	500.03

We extract the unigrams (words) *excluding numbers, hyphens, URLs, separators, punctuations*, and (twitter) *symbols*. Moreover, we convert the words to lower-case, and we exclude the words containing both numbers and letters. We also extract the *profanities*[8] and *non-english* words in the corpuse. Also, we clean non-sense words out the data as much as possible, e.g., the words like zzzzzzz and bbbbb. Furthermore, we keep only the words whose occurrences in the corpus are at least 2. A summary of the results is represented in Table. 3.[9]

Table 3: Unigrams (Words, Profanities, Non-English Words) in the sample Corpus

Words	Unique Words	Profanities	Profanity	Non-English	Non-English
59,683,563	155,108	83,844	0.14 %	98,441	0.16 %

As we see in Table. 3, about 0.14% and 0.16% of the words in the sample corpus is bad and non-English words, respecitvely. We clean the profanities and non-english words out the corpus.

Fig. 2 represents the top 30 most frequenct words in the *clean* corpus with their frequencies.

[8]We have used https://github.com/LDNOOBW/List-of-Dirty-Naughty-Obscene-and-Otherwise-Bad-Words as a reference of profanities, which contains 376 items.

[9]To speed up calculation, we trim the corresponding Document-Feature Matrix (DFM), and tidy them out. Then, we index them. (See corresponding scripts in Appendix.)

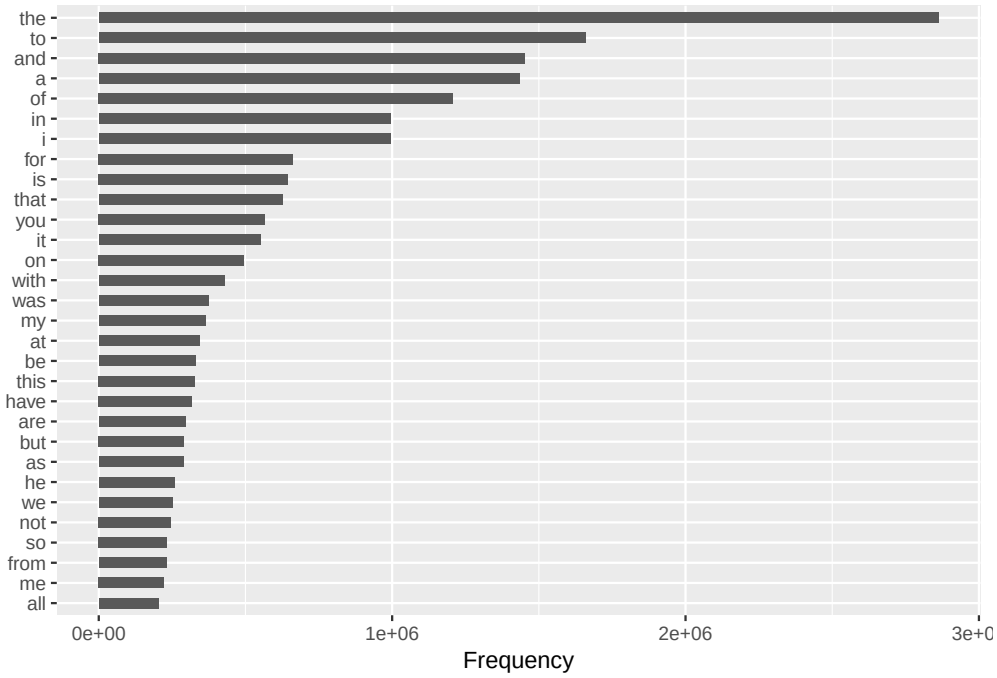

Figure 2: Top 30 most frequent words in the clean corpus

As we see in the above frequency plot, the most frequent words in the sample corpus are *stop-words*. Fig. 3 represents the 30 top frequenct words excluding the stop-words[10] A Word-Cloud for the data, excluding profanities, hyphens, URLs, symbols, stop-words, and numbers, is represented in Fig. 4.

[10]However, note that we do not clean the stop-words out the unigrams.

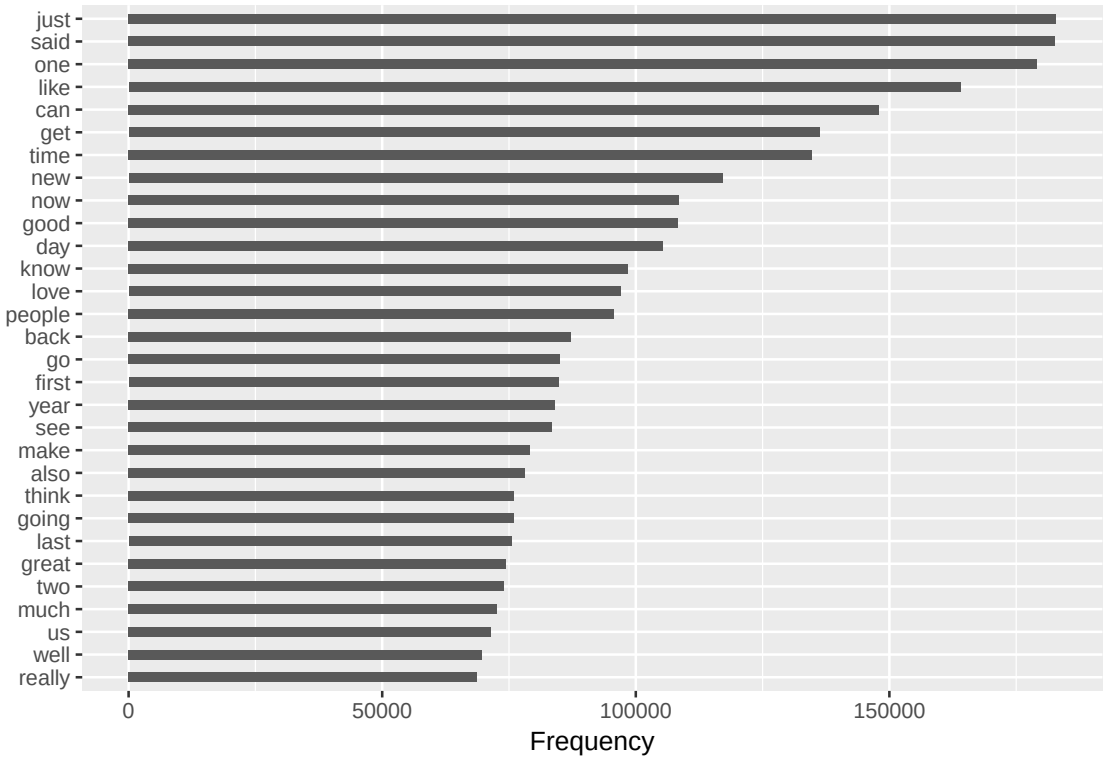

Figure 3: Top 30 most frequent words excluding stopwords

Figure 4: A Word-Cloud for unigrams (excl. stopwords, numbers, profanities, URLs, symbols)

3 Bi-, Tri-, Quad-, and Quint-Grams

In this section, we extract the *bi-grams*, *tri-grams*, *quad-grams*, and *quint-grams* from clean unigrams, and we analyze their frequencies. We keep only those grams that have at least two instances in our corpus. Moreover, we remove those grams which include empty words, or duplicated words, e.g., "a a".

Fig. 5, Fig. 6, Fig. 7, and Fig. 8 represent the 30 most frequenct bi-grams, tri-grams, quad-grams, quint-grams in our corpus, respecitvely.

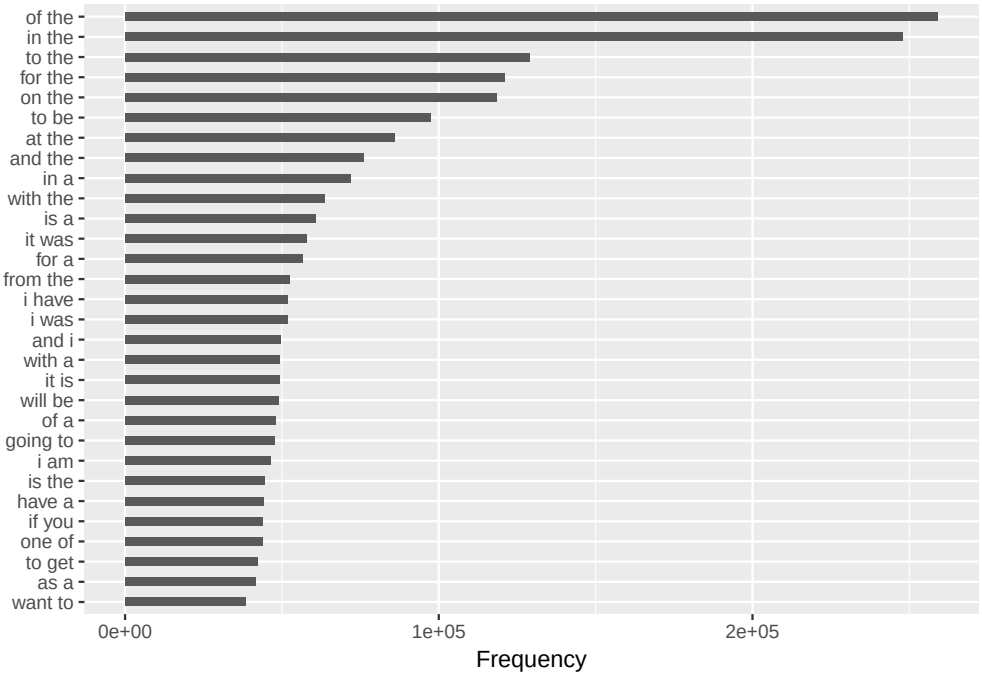

Figure 5: Top 30 most frequent bigrams

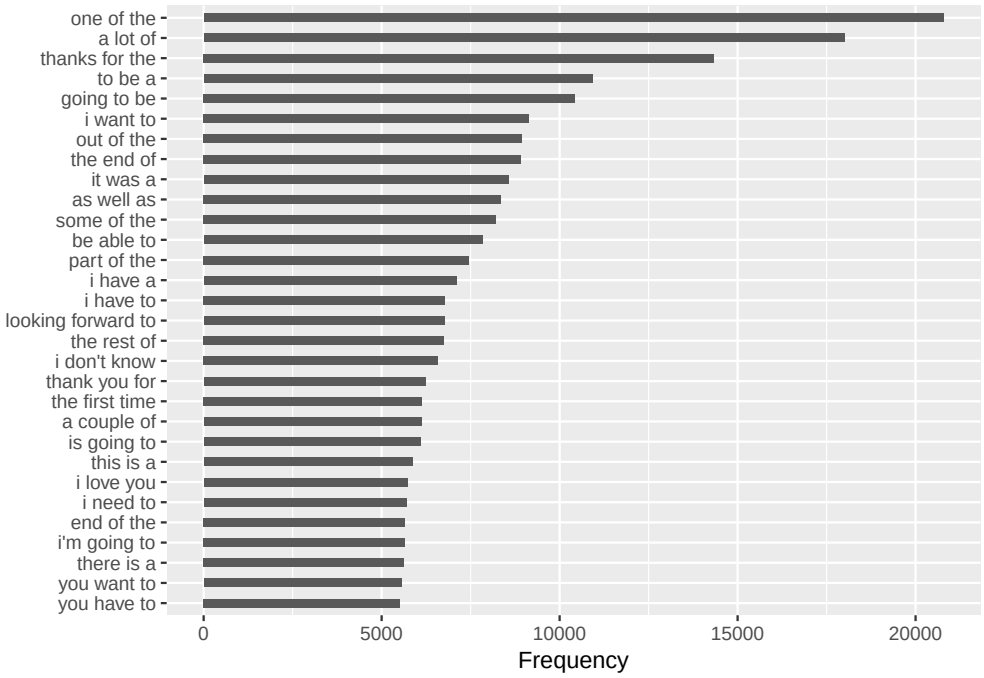

Figure 6: Top 30 most frequent tri-grams

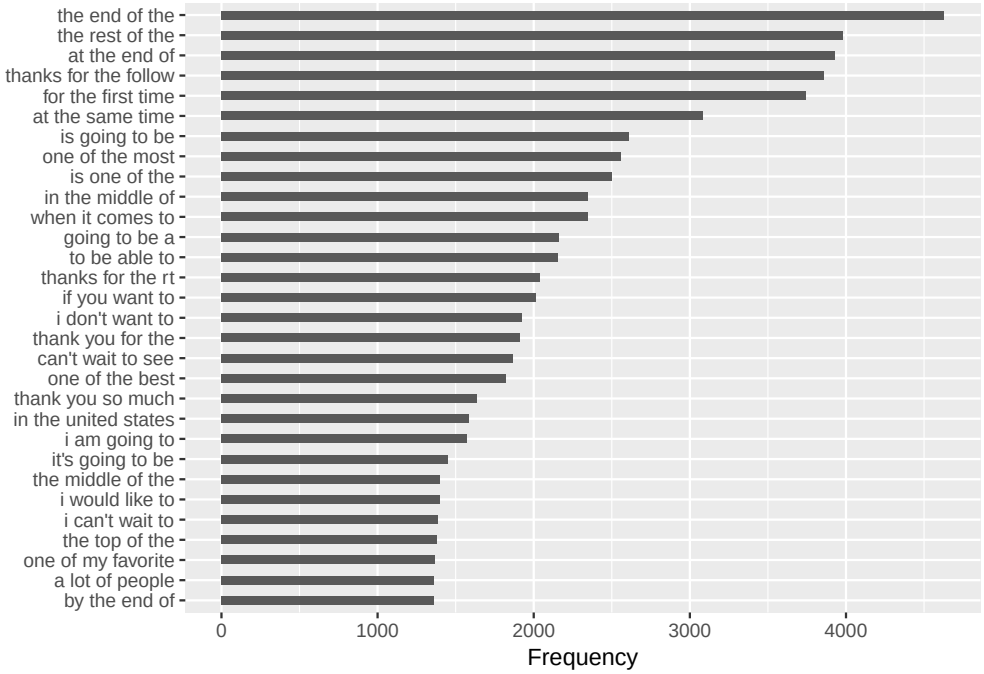

Figure 7: Top 30 most frequent quad-grams

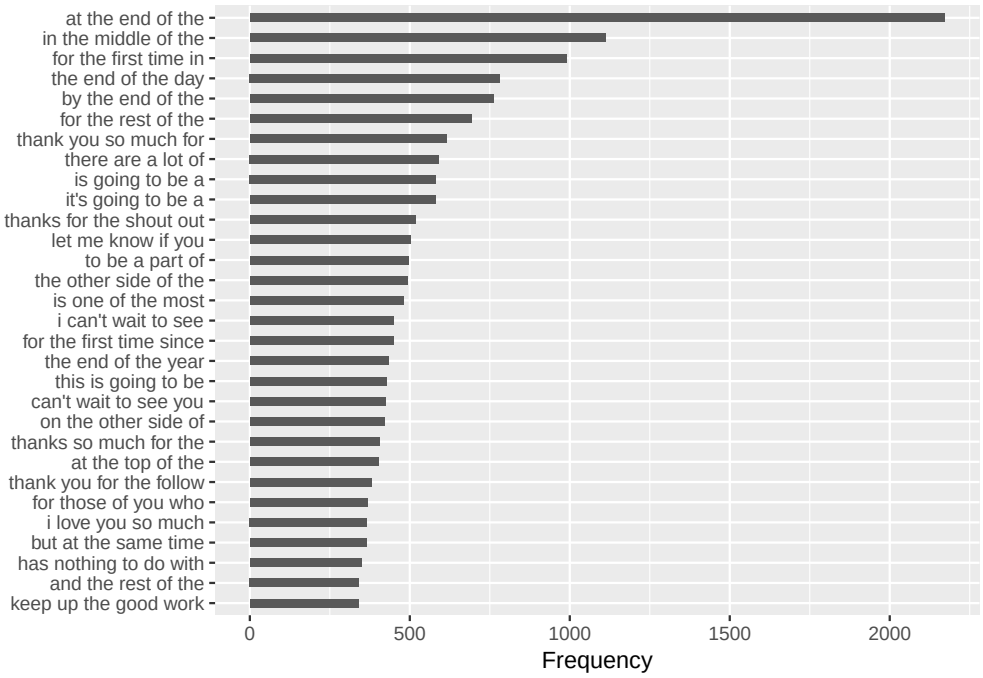

Figure 8: Top 30 most frequent quint-grams

Table. 4 represents the number of terms (grams), the number of instances, the maximum frequency of a term for each n-grams (uni-, bi-, tri-grams). Moreover, it shows how many unique terms we need in a frequency sorted way to cover 50% and 90% of all term instances in the sampled corpus.

Table 4: Coverage Table

	Terms	Instances	Max Frequency	50% Coverage	90% Coverage
Uni-Grams	151,928	59,501,278	2,862,294	81,054	139,743
Bi-Grams	1,701,489	48,442,569	259,091	17,588	583,389
Tri-Grams	2,307,531	25,850,409	20,785	144,428	1,463,788
Quad-Grams	1,261,327	8,911,016	4,630	185,608	964,294
Quint-Grams	448,496	2,456,120	2,172	98,770	366,626

4 Language Modeling

In this section, we descibe how we build our *n-gram model* to predict next words in our application. That is, we are going to model sequences of words using the statistical properties of n-grams. For simplicitiy and practicability, we follow *the Markov assumption*[11] (or independence assumption). That is, in an n-gram model, each word depends only on the last $n-1$ words. This assumption is important because it massively simplifies the problem of estimating the language model from data.

As for probabilities, we use *smoothing* to give a probability to words we have not seen in our training data. There are a few smoothing methods, including *Good-Turing Smoothing*[12] and *Kneser-Ney Smoothing*[13]. We

[11] The markov assumption is that the next state depends only on the current state and is independent of previous history: https://en.wikipedia.org/wiki/Markov_chain

[12] https://en.wikipedia.org/wiki/Good-Turing_frequency_estimation

[13] https://en.wikipedia.org/wiki/Kneser-Ney_smoothing

use the latter, as we think it works better in most cases.[14]

Our product make at most top-5 most likely suggestions to the user. Therefore, it makes sense to keep only the top five n-grams for any given of $n-1$ grams in a n-gram model. This would extremely reduce the memory size we need for our application.

As described in Introduction, we need to find the right balance between size and runtime in order to provide a good experience to the user. We have tried nine different models with two variable factors:

- A fraction of the original corpus (50%, 60%, or 70%)
- An n-gram model (tri-gram, quad-gram, or quint-gram)

We evaluate the quality (*precision*, *average runtime*, and *memory consumption*) of our models by scripts and a testing data provided by Hernán Foffani[15]. The testing data includes two datasets:

- Dataset `blogs` (599 lines, 14587 words)
- Dataset `tweets` (793 lines, 14071 words)

We can find the results of our evaluation in Table. 5. Each row shows the information of the evaluation of a given model, i.e., a row X% fraction & n-Gram for $X \in \{70, 60, 50\}$ and $n \in \{5, 4, 3\}$ denotes an n-Gram model trained on a X% fraction of the original corpus.

Table 5: Model Evaluation

	Top-3 Precision	Precision	Memory Used	Avg Runtime	Size
70% fraction & 5-Gram	21.87%	13.83%	147.91 MB	32.45 msec	40.7 MB
70% fraction & 4-Gram	21.84%	13.68%	131.50 MB	31.61 msec	33.9 MB
70% fraction & 3-Gram	21.20%	12.86%	131.44 MB	26.83 msec	17.4 MB
60% fraction & 5-Gram	21.80%	13.68%	131.39 MB	33.95 msec	35.8 MB
60% fraction & 4-Gram	21.75%	13.56%	109.23 MB	28.97 msec	29.0 MB
60% fraction & 3-Gram	21.25%	12.74%	109.17 MB	31.61 msec	15.2 MB
50% fraction & 5-Gram	21.48%	13.45%	109.12 MB	38.03 msec	29.3 MB
50% fraction & 4-Gram	21.46%	13.37%	147.91 MB	30.40 msec	24.0 MB
50% fraction & 3-Gram	21.06%	12.65%	131.50 MB	30.15 msec	13.0 MB

Note:
'Precision' denotes the top-1 precision.
'Size' denotes the size of the corresponding files.
Since we apply the back-off method, for an n-gram, model, we need to store 1-, ..., n-grams.
'Avg' denotes average.

Since our final product will offer an option to the user to choose the n for an n-gram model (either tri-, quad-, or quint-grams), we should care only about the fraction factor at this point. As we see in the table, the average runtime of the models are all reasonable for this application. The precision of the 50% farction is a little bit lower than the two others. However, there is no a significant difference between precisions (both top-1 and top-3) of the 70% and 60% fractions. Therefore, considering the memory and file sizes, we choose the 60% fraction for our model.

As mentioned earlier, we use the Kneser-Ney proabilities to calculate the probabilities of n-grams. The probability of a given n-gram $w_1 \dots w_n$ is the conditional probability $P(w_n|w_1 \dots w_{n-1})$, i.e., the probability of nth word given the first to $n-1$ words. Table. 6, Table. 7, Table. 8, and Table. 9 represent *the most* and *the least* likely bi-, tri-, quad-, and quint-grams, respectively.

Since our app will return the top-5 most probable word for a given sequence of words, it does not make sense to keep lower-rated n-grams. Moreover, we do not need to keep all unigrams. We just keep top-50 rated

[14]The Kneser-Ney smoothing algorithm has a notion of continuation probability. It also saves us from having to recalculate all our counts using Good-Turing smoothing.
[15]The Git repository at https://github.com/jan-san/dsci-benchmark

Table 6: 10-Most (Left) and 10-Least (Right) Likely Bi-Grams

bigram	Prob	bigram	Prob
specializes in	0.9977663	the yuppies	8e-07
accounted for	0.9977102	the zafars	8e-07
gearing up	0.9975610	the zagg	8e-07
reminiscent of	0.9975600	the zapruder	8e-07
according to	0.9971085	the zilgans	8e-07
mardi gras	0.9966960	the zimmermans	8e-07
stainless steel	0.9959018	the zirconium	8e-07
cinco de	0.9957293	the zohan	8e-07
specializing in	0.9956632	the zohar	8e-07
foie gras	0.9954546	the zouave	8e-07

Table 7: 10-Most (Left) and 10-Least (Right) Likely Tri-Grams

trigram	Prob	trigram	Prob
was able to	0.9999958	of the greedy	9.5e-06
been able to	0.9999940	of the hm	9.5e-06
year according to	0.9999926	of the receiving	9.5e-06
in according to	0.9999916	of the climategate	9.5e-06
were able to	0.9999914	of the digestive	9.5e-06
was supposed to	0.9999907	of the outlawz	9.5e-06
and according to	0.9999888	of the racks	9.5e-06
is supposed to	0.9999869	of the rue	9.5e-06
the midst of	0.9999826	of the sleepy	9.5e-06
happy cinco de	0.9999814	of the sophomore	9.5e-06

Table 8: 10-Most (Left) and 10-Least (Right) Likely Quad-Grams

quadgram	Prob	quadgram	Prob
i was able to	1	one of the shorter	0.0001306
have been able to	1	one of the legal	0.0001306
he was able to	1	one of the rings	0.0001306
we were able to	1	one of the victim's	0.0001306
in the midst of	1	one of the nominees	0.0001306
haven't been able to	1	one of the drawbacks	0.0001306
i was supposed to	1	one of the moms	0.0001305
not been able to	1	one of the giant	0.0001305
and was able to	1	one of the churches	0.0001305
she was able to	1	one of the editors	0.0001305

unigrams. For example, Table. 10, Table. 11, Table. 12, and Table. 13 present the top-5 bi-, tri-, quad-, quint-grams starting with "a", "a lot", "a lot of", and "a lot of people", respectively.

Table 9: 10-Most (Left) and 10-Least (Right) Likely Quint-Grams

quintgram	Prob	quintgram	Prob
and i was able to	1	the end of the there	0.0006037
that i was able to	1	the end of the nineteenth	0.0006037
but i was able to	1	the end of the study	0.0006037
so i was able to	1	the end of the set	0.0006036
glad i was able to	1	the end of the s	0.0006036
i haven't been able to	1	the end of the pier	0.0006036
i have been able to	1	the end of the party	0.0006036
because i was able to	1	the end of the lane	0.0006036
would have been able to	1	the end of the exit	0.0006036
is in the midst of	1	the end of the final	0.0006036

Table 10: Top-5 Bi-Grams starting with 'a'

word1	word2	count	Prob
a	lot	28648	0.0204716
a	few	26354	0.0188328
a	little	24559	0.0175528
a	great	24376	0.0174239
a	good	23580	0.0168570

Table 11: Top-5 Tri-Grams starting with 'a lot'

word1	word2	word3	count	Prob
a	lot	of	18006	0.6726562
a	lot	more	1534	0.0572874
a	lot	to	1053	0.0393222
a	lot	about	610	0.0227601
a	lot	and	362	0.0135271

Table 12: Top-5 Tri-Grams starting with 'a lot of'

word1	word2	word3	word4	count	Prob
a	lot	of	people	1364	0.0997636
a	lot	of	information	51	0.0501175
a	lot	of	thought	40	0.0488378
a	lot	of	effort	31	0.0487776
a	lot	of	the	657	0.0486688

Table 13: Top-5 Quint-Grams starting with 'a lot of people'

word1	word2	word3	word4	word5	count	Prob
a	lot	of	people	are	105	0.1110059
a	lot	of	people	who	104	0.1105242
a	lot	of	people	in	77	0.0814335
a	lot	of	people	have	59	0.0621468
a	lot	of	people	don't	43	0.0449770

Scripts

Packages & Functions

```
#----- Required Libraries ----#

require(quanteda)
require(readr)
require(stopwords)
require(tidytext)
require(dplyr)
require(ngram)
require(knitr)
require(kableExtra)
require(xtable)
require(stringr)
require(ggplot2)
require(data.table)
```

```
#---- Some Auxiliary Functions -----#

'%!in%' <- function(x,y)!('%in%'(x,y))

# find the coverage
findMany <- function(dt, p){
        cri <- sum(dt$count) * p
        com <- 0
        ind <- 0
        for(i in 1:dim(dt)[1]){
                com <- com + dt$count[i]
                if(com >= cri){
                        ind <- i
                        break
                }
        }
        ind
}
```

Preprocessing & Uni-Grams

```
#=========================================#

# Loading the Data (Blogs, Twitter, News)

#=========================================#


blogs <- read_lines("rawData/blogs.txt")
twitter <- readLines("rawData/twitter.txt", skipNul = TRUE)
news <- read_lines("rawData/news.txt")
```

```r
#=========================================#

# Getting Some Information about the Raw Data

#=========================================#


size_blogs <- round((file.info("rawData/blogs.txt")$size)/1000000, 2)
size_twitter <- round((file.info("rawData/twitter.txt")$size)/1000000, 2)
size_news <- round((file.info("rawData/news.txt")$size)/1000000, 2)
size_corpus <- size_blogs + size_twitter + size_news

lines_blogs <- length(blogs)
lines_twitter <- length(twitter)
lines_news <- length(news)
lines_corpus <- lines_blogs + lines_twitter + lines_news

words_blogs <- format(wordcount(blogs),big.mark=",",scientific=FALSE)
words_twitter <- format(wordcount(twitter),big.mark=",",scientific=FALSE)
words_news <- format(wordcount(news),big.mark=",",scientific=FALSE)
words_corpus <- format(wordcount(blogs)+wordcount(twitter)+wordcount(news),
                       big.mark=",",scientific=FALSE)


length_blogs <- sapply(blogs, nchar)
max_length_blogs <- max(length_blogs)
max_length_blogs <- format(max_length_blogs,big.mark=",",scientific=FALSE)
min_length_blogs <- min(length_blogs)
min_max_chars_blogs <- paste(as.character(min_length_blogs), " - ",
                             as.character(max_length_blogs), sep = "")

length_twitter <- sapply(twitter, nchar)
max_length_twitter <- max(length_twitter)
max_length_twitter <- format(max_length_twitter,big.mark=",",scientific=FALSE)
min_length_twitter <- min(length_twitter)
min_max_chars_twitter <- paste(as.character(min_length_twitter), " - ",
                               as.character(max_length_twitter), sep = "")

length_news <- sapply(news, nchar)
max_length_news <- max(length_news)
max_length_news <- format(max_length_news,big.mark=",",scientific=FALSE)
min_length_news <- min(length_news)
min_max_chars_news <- paste(as.character(min_length_news), " - ",
                            as.character(max_length_news), sep = "")


max_length_corpus <- max(max_length_news, max_length_blogs, max_length_twitter)
max_length_corpus <- format(max_length_corpus,big.mark=",",scientific=FALSE)
min_length_corpus <- min(min_length_news, min_length_blogs, min_length_twitter)
min_max_chars_corpus <- paste(as.character(min_length_corpus), " - ",
                            as.character(max_length_corpus), sep = "")

avg_nchar_news <- round(mean(length_news), 2)
```

```
avg_nchar_twitter <- round(mean(length_twitter), 2)
avg_nchar_blogs <- round(mean(length_blogs), 2)
avg_nchar_corpus <- avg_nchar_news + avg_nchar_blogs + avg_nchar_twitter


info_blogs <- c("Blogs", paste(as.character(size_blogs), "MB"),
                format(lines_blogs,big.mark=",",scientific=FALSE),
                as.character(words_blogs),
                min_max_chars_blogs,
                avg_nchar_blogs)

info_twitter <- c("Twitter", paste(as.character(size_twitter), "MB"),
                  format(lines_twitter,big.mark=",",scientific=FALSE),
                  words_twitter,
                  min_max_chars_twitter,
                  avg_nchar_twitter)

info_news <- c("News", paste(as.character(size_news), "MB"),
               format(lines_news,big.mark=",",scientific=FALSE),
               words_news,
               min_max_chars_news,
               avg_nchar_news)

info_corpus <- c("Corpus", paste(as.character(size_corpus), "MB"),
                 format(lines_corpus,big.mark=",",scientific=FALSE),
                 words_corpus,
                 min_max_chars_corpus,
                 avg_nchar_corpus)

raw_info <- as.data.frame(rbind(info_blogs, info_twitter, info_news, info_corpus))
colnames(raw_info) = c("Data",
                       "Size",
                       "Lines",
                       "Words",
                       "Range nchars",
                       "Avg nchars")
rownames(raw_info) = NULL



# The Table

kable(raw_info, "latex", booktabs = T,
      caption = "Raw Data - Information", align = "c") %>%
        kable_styling(latex_options = "hold_position") %>%
        footnote(general =  c("Words: approximate in million",
                              "Range nchars: range of number of chars in lines",
                              "Avg nchars: avgerage number of chars in lines"))
```

```
#=========================================#

# Sampling the Data & Making a Corpus

#=========================================#
```

```r
set.seed(2019)
blogs_ind <- sample(length(blogs), length(blogs) * 0.6)
twitter_ind <- sample(length(twitter), length(twitter) * 0.6)
news_ind <- sample(length(news), length(news) * 0.6)

blogs_sample <- blogs[blogs_ind]
twitter_sample <- twitter[twitter_ind]
news_sample <- news[news_ind]

sample_corpus <- corpus(c(blogs_sample,
                          twitter_sample,
                          news_sample))
```

```r
#=========================================#

# Raw Sampled data info

#=========================================#


lines_blogs_sample <- length(blogs_sample)
lines_twitter_sample <- length(twitter_sample)
lines_news_sample <- length(news_sample)
lines_corpus_sample <- lines_blogs_sample +
        lines_twitter_sample +
        lines_news_sample

words_blogs_sample <- format(wordcount(blogs_sample),
                             big.mark=",",
                             scientific=FALSE)
words_twitter_sample <- format(wordcount(twitter_sample),
                               big.mark=",",
                               scientific=FALSE)
words_news_sample <- format(wordcount(news_sample),
                            big.mark=",",
                            scientific=FALSE)
words_corpus_sample <- format(wordcount(blogs_sample) +
                                      wordcount(twitter) +
                                      wordcount(news),
                      big.mark=",",
                      scientific=FALSE)



length_blogs_sample <- sapply(blogs_sample, nchar)
range_chars_blogs_sample <- paste(as.character(min(length_blogs_sample)), " - ",
                             as.character(max(length_blogs_sample)), sep = "")

length_twitter_sample <- sapply(twitter_sample, nchar)
range_chars_twitter_sample <- paste(as.character(min(length_twitter_sample)), " - ",
                               as.character(max(length_twitter_sample)), sep = "")
```

```
length_news_sample <- sapply(news_sample, nchar)
range_chars_news_sample <- paste(as.character(min(length_news_sample)), " - ",
                            as.character(max(length_news_sample)), sep = "")

min_sample_corp <- min(min(length_blogs_sample),
                       min(length_twitter_sample),
                       min(length_news_sample))

max_sample_corp <- max(max(length_blogs_sample),
                       max(length_twitter_sample),
                       max(length_news_sample))

range_chars_corpus_sample <- paste(as.character(min_sample_corp), " - ",
                            as.character(max_sample_corp), sep = "")


avg_nchar_news_sample <- round(mean(length_news_sample), 2)
avg_nchar_twitter_sample <- round(mean(length_twitter_sample), 2)
avg_nchar_blogs_sample <- round(mean(length_blogs_sample), 2)
avg_nchar_corpus_sample <- avg_nchar_news_sample +
        avg_nchar_twitter_sample +
        avg_nchar_blogs_sample

info_blogs_sample <- c("Blogs",
                format(lines_blogs_sample,big.mark=",",scientific=FALSE),
                words_blogs_sample,
                range_chars_blogs_sample,
                avg_nchar_blogs_sample)

info_twitter_sample <- c("Twitter",
                  format(lines_twitter_sample,big.mark=",",scientific=FALSE),
                  words_twitter_sample,
                  range_chars_twitter_sample,
                  avg_nchar_twitter_sample)

info_news_sample <- c("News",
               format(lines_news_sample,big.mark=",",scientific=FALSE),
               words_news_sample,
               range_chars_news_sample,
               avg_nchar_news_sample)

info_corpus_sample <- c("Corpus",
                format(lines_corpus_sample,big.mark=",",scientific=FALSE),
               words_corpus_sample,
               range_chars_corpus_sample,
               avg_nchar_corpus_sample)

raw_info_sample <- as.data.frame(rbind(info_blogs_sample,
                                       info_twitter_sample,
                                       info_news_sample,
                                       info_corpus_sample))
colnames(raw_info_sample) = c("Data",
                        "Lines",
```

```
                        "Words",
                        "Range nchars",
                        "Avg nchars")
rownames(raw_info_sample) = NULL

kable(raw_info_sample, "latex", booktabs = T,
      caption = "Random Sampled Raw Data - Information",
       align = "c") %>%
        kable_styling(latex_options = "hold_position")
```

```
#=========================================#

# Uni-Grams

#=========================================#


unigrams <- tokens(x = tolower(sample_corpus),
                        remove_numbers = TRUE,
                        remove_hyphens = TRUE,
                        remove_url = TRUE,
                        remove_symbols = TRUE,
                        remove_separators = TRUE,
                        remove_punct = TRUE,
                        remove_twitter = TRUE)

# Remove words containing numbers
unigrams <- tokens_remove(unigrams,
                          pattern = "\\w*[0-9]+\\w*\\s*",
                          valuetype = "regex")


pat <- "\\w*(kk|nn|zz|aaa|
bbb|ccc|ddd|eee|fff|ggg|hhh|jjj|lll|ppp|qqq|rrr|vvv|xxx|
iiii|ssss|wwww|mmmmm|ooooo|ttttt|uuuuu|yyyyy)+\\w*\\s*"

# Remove nosnese words like zzz
unigrams <- tokens_remove(unigrams,
                          pattern = pat,
                          valuetype = "regex")
```

```
#=========================================#

#  DFM of Unigrams

#=========================================#


dfm_uni <- dfm(unigrams)
dfm_uni <- dfm_trim(dfm_uni, min_termfreq = 3)
```

```
#=========================================#

#  Unigrams - Tidy Data Table
```

```r
#=========================================#


dt_uni <- tidy(dfm_uni)
setDT(dt_uni)
dt_uni <- dt_uni[, document := NULL][, .(word = term, count)]
dt_uni <- dt_uni[, .(count = sum(count)), by = word]
setkey(dt_uni, word)
```

```r
#=========================================#

#  Profanities

#=========================================#


profanities <- read_lines("rawData/bad_words.txt")
dfm_profanities <- dfm_select(dfm_uni, pattern = profanities,
                          selection = "keep")

dt_profanities <- tidy(dfm_profanities)
setDT(dt_profanities)
dt_profanities <- dt_profanities[, document := NULL][, .(word = term, count)]
dt_profanities <- dt_profanities[, .(count = sum(count)), by = word]
setkey(dt_profanities, word)
```

```r
#=========================================#

#  Non-English Words

#=========================================#


noneng_ind <- grepl("[^\x01-\x7F]+", dt_uni$word)
# A data table with non-english words
dt_uni_neng <- dt_uni[which(noneng_ind), ]
```

```r
#=========================================#

#  Information Table for Unigrams

#=========================================#


# nummber of word instances in the corpus
num_words <- sum(dt_uni$count)

# number of words in the corpus
num_uniq_words <- format(length(unique(dt_uni$word)),
                    big.mark=",",
                    scientific=FALSE)


# nummber of profanity instances in the corpus
```

```r
num_profanities <- sum(dt_profanities$count)

# Portion - profanities in the corpus
profanities_perc <- round(num_profanities / num_words * 100, 2)


# number of non english instanes
num_neng_words <- sum(dt_uni_neng$count)

# Portion - non english words in the corpus
neng_perc <- round((num_neng_words / num_words) * 100, 2)



num_words <- format(sum(dt_uni$count),
                    big.mark=",",
                    scientific=FALSE)

num_profanities <- format(sum(dt_profanities$count),
                    big.mark=",",
                    scientific=FALSE)

num_neng_words <- format(sum(dt_uni_neng$count),
                    big.mark=",",
                    scientific=FALSE)

info_uni <- data.frame(num_words,
                       num_uniq_words,
                       num_profanities,
                       paste(as.character(profanities_perc), "%"),
                       num_neng_words,
                       paste(as.character(neng_perc), "%"))


colnames(info_uni) <- c("Words",
                        "Unique Words",
                        "Profanities",
                        "Profanity",
                        "Non-English",
                        "Non-English")


kable(info_uni, "latex", booktabs = T,
      caption = "Unigrams (Words, Profanities, Non-English Words)
      in the sample Corpus",
      align = "c") %>%
        kable_styling(latex_options = "hold_position")
```

```r
#=========================================#

#  Profanity & Non-English Filtering

#=========================================#
```

```
unigrams <- tokens_remove(unigrams, pattern = profanities)

unigrams <- tokens_remove(unigrams,
                                pattern = "[A-z]*[^\x01-\x7F]+[A-z]*",
                                valuetype = "regex")
```

```
#=========================================#

#  DFM of Unigrams  - Clean

#=========================================#


dfm_uni <- dfm(unigrams)
dfm_uni <- dfm_trim(dfm_uni, min_termfreq = 3)
```

```
#=========================================#

#  Unigrams - Tidy Data Table - Clean

#=========================================#


dt_uni <- tidy(dfm_uni)
setDT(dt_uni)
dt_uni <- dt_uni[, document := NULL][, .(word = str_squish(term), count)]
dt_uni <- dt_uni[, .(count = sum(count)), by = word]
setkey(dt_uni, word)
```

```
#=========================================#

#  Frequency Plot - Unigrams

#=========================================#

ggplot(dt_uni[order(-count)][1:30, ],
       aes(x = reorder(word, count),
           y = count))  +
        geom_col(width = 0.5) +
        xlab(NULL) +
        ylab("Frequency") +
        coord_flip()
```

```
#=========================================#

#  DFM & Table for Unigrams excluding Stop-Words

#=========================================#


dfm_uni_wostp <- dfm_remove(dfm_uni, stopwords("english"))

dt_uni_wostp <- tidy(dfm_uni_wostp)
setDT(dt_uni_wostp)
dt_uni_wostp <- dt_uni_wostp[, document := NULL][, .(word = term, count)]
```

```
dt_uni_wostp <- dt_uni_wostp[, .(count = sum(count)), by = word]
setkey(dt_uni_wostp, word)
dt_uni_wostp <- dt_uni_wostp[order(-count)]
```

```
#=========================================#

#  Frequency Plot - Unigrams excluding Stop-Words

#=========================================#

ggplot(dt_uni_wostp[1:30, ],
       aes(x = reorder(word, count),
           y = count))  +
        geom_col(width = 0.4) +
        xlab(NULL) +
        ylab("Frequency") +
        coord_flip()
```

```
#=========================================#

#  Worldcloud - Unigrams without Stop-Words

#=========================================#


set.seed(1000)
textplot_wordcloud(dfm_uni_wostp, random_order = FALSE,
                   rotation = .25,
                   color = RColorBrewer::brewer.pal(8,"Dark2")
                  )
```

Bi-, Tri-, Quad-, & Quint-Grams

```
#=========================================#

# Bi-Grams

#=========================================#

bigrams <- tokens_ngrams(unigrams, n = 2)
```

```
#=========================================#

# DFM for Bi-Grams

#=========================================#

dfm_bi <- dfm(bigrams)
dfm_bi <- dfm_trim(dfm_bi, min_termfreq = 3)
```

```
#=========================================#

# DFM for Bi-Grams
```

```
#=========================================#

dt_bi <- tidy(dfm_bi)
setDT(dt_bi)
dt_bi <- dt_bi[, document := NULL][, .(count = sum(count)), by = term]
dt_bi[, c("word1", "word2") := tstrsplit(term, "_", fixed=TRUE)]
dt_bi <- dt_bi[, .(word1, word2, count)]

dt_bi <- dt_bi[word1 != ""][word2 != ""]
dt_bi <- dt_bi[(word1 != word2)]

setkey(dt_bi, word1, word2)
dt_bi <- dt_bi[order(-count)]
```

```
#=========================================#

# Frequency Plot for Bi-Grams

#=========================================#


ggplot(dt_bi[1:30, ],
       aes(x = reorder(paste(word1, word2, sep = " "), count),
           y = count))  +
        geom_col(width = 0.4) +
        xlab(NULL) +
        ylab("Frequency") +
        coord_flip()
```

```
#=========================================#

# Tri-Grams

#=========================================#


trigrams <- tokens_ngrams(unigrams, n = 3)
```

```
#=========================================#

# DFM for Tri-Grams

#=========================================#

dfm_tri <- dfm(trigrams)
dfm_tri <- dfm_trim(dfm_tri, min_termfreq = 3)
```

```
#=========================================#

# Tidy Data Table for Tri-Grams

#=========================================#
```

```
dt_tri <- tidy(dfm_tri)
setDT(dt_tri)
dt_tri <- dt_tri[, document := NULL][, .(count = sum(count)), by = term]
dt_tri[, c("word1", "word2", "word3") := tstrsplit(term, "_", fixed=TRUE)]
dt_tri <- dt_tri[, .(word1, word2, word3, count)]

dt_tri <- dt_tri[!((word1 == word2) | (word2 == word3))]
dt_tri <- dt_tri[word1!="" & word2!="" & word3!=""]

setkey(dt_tri, word1, word2, word3)
dt_tri <- dt_tri[order(-count)]
```

```
#=========================================#

# Frequency Plot for Tri-Grams

#=========================================#

ggplot(dt_tri[1:30, ],
       aes(x = reorder(paste(word1, word2, word3, sep = " "), count),
           y = count))  +
        geom_col(width = 0.4) +
        xlab(NULL) +
        ylab("Frequency") +
        coord_flip()
```

```
#=========================================#

# Quad-Grams

#=========================================#

quadgrams <- tokens_ngrams(unigrams, n = 4)
```

```
#=========================================#

# DFM for Quad-Grams

#=========================================#

dfm_quad <- dfm(quadgrams)
dfm_quad <- dfm_trim(dfm_quad, min_termfreq = 3)
```

```
#=========================================#

# Tidy Data Table for Quad-Grams

#=========================================#


dt_quad <- tidy(dfm_quad)
setDT(dt_quad)
dt_quad <- dt_quad[, document := NULL][, .(count = sum(count)), by = term]
dt_quad[, c("word1", "word2", "word3", "word4") :=
               tstrsplit(term, "_", fixed=TRUE)]
```

```
dt_quad <- dt_quad[, .(word1, word2, word3, word4, count)]

dt_quad <- dt_quad[!((word1 == word2) | (word2 == word3) | (word3 == word4))]
dt_quad <- dt_quad[word1!="" & word2!="" & word3!="" & word4!=""]

setkey(dt_quad, word1, word2, word3, word4)
dt_quad <- dt_quad[order(-count)]
```

```
#=========================================#

# Frequency Plot for Quad-Grams

#=========================================#

ggplot(dt_quad[1:30, ],
       aes(x = reorder(paste(word1, word2, word3, word4, sep = " "), count),
           y = count))  +
        geom_col(width = 0.4) +
        xlab(NULL) +
        ylab("Frequency") +
        coord_flip()
```

```
#=========================================#

# Quint-Grams

#=========================================#

quintgrams <- tokens_ngrams(unigrams, n = 5)
```

```
#=========================================#

# DFM for Quint-Grams

#=========================================#

dfm_quint <- dfm(quintgrams)
dfm_quint <- dfm_trim(dfm_quint, min_termfreq = 3)
```

```
#=========================================#

# Tidy Data Table for Quint-Grams

#=========================================#

dt_quint <- tidy(dfm_quint)
setDT(dt_quint)
dt_quint <- dt_quint[, document := NULL][, .(count = sum(count)), by = term]
dt_quint[, c("word1", "word2", "word3", "word4", "word5") :=
               tstrsplit(term, "_", fixed=TRUE)]
dt_quint <- dt_quint[, .(word1, word2, word3, word4, word5, count)]

dt_quint <- dt_quint[!((word1 == word2) | (word2 == word3) |
                           (word3 == word4) | (word4 == word5) )]
dt_quint <- dt_quint[word1!="" & word2!="" & word3!="" & word4!="" & word5!=""]
```

```r
setkey(dt_quint, word1, word2, word3, word4, word5)
dt_quint <- dt_quint[order(-count)]
```

```r
#=========================================#

# Frequency Plot for Quint-Grams

#=========================================#

ggplot(dt_quint[1:30, ],
       aes(x = reorder(paste(word1, word2, word3, word4, word5, sep = " "), count),
           y = count))  +
        geom_col(width = 0.4) +
        xlab(NULL) +
        ylab("Frequency") +
        coord_flip()
```

```r
#=========================================#

# General Information about all N-grams

#=========================================#

dt_coverage <- data.frame(
        Terms = c(format(nrow(dt_uni),big.mark=",",scientific=FALSE),
                  format(nrow(dt_bi),big.mark=",",scientific=FALSE),
                  format(nrow(dt_tri),big.mark=",",scientific=FALSE),
                  format(nrow(dt_quad),big.mark=",",scientific=FALSE),
                  format(nrow(dt_quint),big.mark=",",scientific=FALSE)
                  ),
        Instances = c(format(sum(dt_uni$count),big.mark=",",scientific=FALSE),
                      format(sum(dt_bi$count),big.mark=",",scientific=FALSE),
                      format(sum(dt_tri$count),big.mark=",",scientific=FALSE),
                      format(sum(dt_quad$count),big.mark=",",scientific=FALSE),
                      format(sum(dt_quint$count),big.mark=",",scientific=FALSE)
                      ),
        Max_Freq = c(format(max(dt_uni$count),big.mark=",",scientific=FALSE),
                     format(max(dt_bi$count),big.mark=",",scientific=FALSE),
                     format(max(dt_tri$count),big.mark=",",scientific=FALSE),
                     format(max(dt_quad$count),big.mark=",",scientific=FALSE),
                     format(max(dt_quint$count),big.mark=",",scientific=FALSE)
                     ),
        Cov_50 = c(format(findMany(dt_uni, .5),big.mark=",",scientific=FALSE),
                   format(findMany(dt_bi, .5),big.mark=",",scientific=FALSE),
                   format(findMany(dt_tri, .5),big.mark=",",scientific=FALSE),
                   format(findMany(dt_quad, .5),big.mark=",",scientific=FALSE),
                   format(findMany(dt_quint, .5),big.mark=",",scientific=FALSE)
                   ),
        Cov_90 = c(format(findMany(dt_uni, .9),big.mark=",",scientific=FALSE),
                   format(findMany(dt_bi, .9),big.mark=",",scientific=FALSE),
                   format(findMany(dt_tri, .9),big.mark=",",scientific=FALSE),
                   format(findMany(dt_quad, .9),big.mark=",",scientific=FALSE),
                   format(findMany(dt_quint, .9),big.mark=",",scientific=FALSE)
                   )
```

```
)

colnames(dt_coverage) = c("Terms", "Instances", "Max Frequency",
                          "50% Coverage", "90% Coverage")

rownames(dt_coverage) = c("Uni-Grams", "Bi-Grams", "Tri-Grams",
                          "Quad-Grams", "Quint-Grams")


kable(dt_coverage, "latex", booktabs = T,
      caption = "Coverage Table", align = "c") %>%
        kable_styling(latex_options = "hold_position")
```

The Model

```
#=========================================#

# Models Evaluation

#=========================================#


precision <- c("21.87%", "21.84%", "21.20%", "21.80%",
            "21.75%", "21.25%", "21.48%", "21.46%", "21.06%")

precision1 <- c("13.83%", "13.68%", "12.86%", "13.68%",
            "13.56%", "12.74%", "13.45%", "13.37%", "12.65%")

memory <- c("147.91 MB", "131.50 MB", "131.44 MB",
            "131.39 MB", "109.23 MB", "109.17 MB", "109.12 MB")

runtime <- c("32.45 msec", "31.61 msec", "26.83 msec",
             "33.95 msec", "28.97 msec", "31.61 msec",
             "38.03 msec", "30.40 msec", "30.15 msec")

size <- c("40.7 MB", "33.9 MB", "17.4 MB",
          "35.8 MB", "29.0 MB", "15.2 MB",
          "29.3 MB", "24.0 MB", "13.0 MB")

eval_data <- data.frame(cbind(precision, precision1,
                              memory, runtime, size))

rownames(eval_data) <- c("70% fraction & 5-Gram", "70% fraction & 4-Gram",
                         "70% fraction & 3-Gram", "60% fraction & 5-Gram",
                         "60% fraction & 4-Gram", "60% fraction & 3-Gram",
                         "50% fraction & 5-Gram", "50% fraction & 4-Gram",
                         "50% fraction & 3-Gram")

colnames(eval_data) <- c("Top-3 Precision", "Precision", "Memory Used",
                         "Avg Runtime", "Size")

x <- paste("Since we apply the back-off method, for an n-gram,",
```

```
          "model, we need to store 1-, ...., n-grams.")
kable(eval_data, "latex", booktabs = T,
      caption = "Model Evaluation", align = "c") %>%
        kable_styling(latex_options = "hold_position") %>%
        footnote(general =  c("`Precision` denotes the top-1 precision.",
                              "`Size` denotes the size of the corresponding files.",
                              x,
                              "`Avg` denotes average."))
```

```
#=========================================#

# Uni- and BI-Grams Proabability

#=========================================#

# Discount Weigth
d <- 0.75

# Total Number of Bigrams
total_bi <- nrow(dt_bi)

# Num of occurrences of bigrams in the corpus as first word
count_Biw1 <- dt_bi[ , .(count_Biw1 = sum(count)), by = word1]
setkey(count_Biw1, word1)
dt_bi[, count_w1 := count_Biw1[word1, count_Biw1]]

# Num of unique bigrams with w1 as their first word
num_Biw1 <- dt_bi[, .(num = .N), by = word1]
setkey(num_Biw1, word1)
dt_bi[, num_w1 := num_Biw1[word1, num]]

# Num of unique bigrams with w2 as their second word
num_Biw2 <- dt_bi[, .(num = .N), by = word2]
setkey(num_Biw2, word2)
dt_bi[, num_w2 := num_Biw2[word2, num]]


# Uni-Gram Probability
dt_bi[, prob_w2 := num_w2/total_bi]
prob_uni <- unique(dt_bi[, .(word = word2, Prob = prob_w2)])
setkey(prob_uni, word)
dt_uni[, Prob := prob_uni[word, Prob]]
dt_uni <- dt_uni[!is.na(Prob)]
dt_uni <- dt_uni[order(-Prob)]

# Bi-Gram Probability
dt_bi[, Prob := ((count - d)/ count_w1) +
              ((d/count_w1) * num_w1 * prob_w2)]

dt_bi[, count_w1:=NULL][,num_w1:=NULL][,num_w2:=NULL][,prob_w2:=NULL]
dt_bi <- dt_bi[order(-Prob)]
```

```
#=========================================#
```

```
# Tri-Grams Proabability

#=========================================#


# Num of occurrences of trigrams in the corpus as 1st,2nd words
count_Triw1w2 <- dt_tri[ , .(count = sum(count)), by = .(word1, word2)]
count_Triw1w2[, term := paste(word1, word2)]
count_Triw1w2[, word1 := NULL][, word2 := NULL]
setkey(count_Triw1w2, term)
dt_tri[, term := paste(word1, word2)]
dt_tri[, count_w1w2 := count_Triw1w2[term, count]]

# Num of unique trigrams with as 1st,2nd words
num_Triw1w2 <- dt_tri[, .(num = .N), by = .(word1, word2)]
num_Triw1w2[, term := paste(word1, word2)]
num_Triw1w2[, word1 := NULL][, word2 := NULL]
setkey(num_Triw1w2, term)
dt_tri[, num_w1w2 := num_Triw1w2[term, num]]

# Normalizing Constant
dt_tri[, lambda_w1w2 := (d/count_w1w2)*num_w1w2]

# Tri-Gram Probability
bi_temp <- dt_bi[, term := paste(word1, word2)]
setkey(bi_temp, term)
dt_tri[, term2 := paste(word2, word3)]
dt_tri[, Prob := ((count - d)/count_w1w2) +
                         (lambda_w1w2 * bi_temp[term2, Prob])]
dt_bi[, term := NULL]
dt_tri <- dt_tri[, .(word1, word2, word3, count, Prob)]
dt_tri <- dt_tri[!is.na(Prob)]
dt_tri <- dt_tri[order(-Prob)]
```

```
#=========================================#

# Most and Least Likely Bi-Grams

#=========================================#

t1 <- head(dt_bi[order(-Prob)], 10)
t1 <- t1[, bigram := paste(word1, word2)][, .(bigram, Prob)]

t2 <- tail(dt_bi[order(-Prob)], 10)
t2 <- t2[, bigram := paste(word1, word2)][, .(bigram, Prob)]


kable(list(t1, t2), "latex", booktabs = T,
      caption = "10-Most (Left) and 10-Least (Right) Likely Bi-Grams", align = "c") %>%
        kable_styling(latex_options = "hold_position")
```

```
#=========================================#

# Most and Least Likely Tri-Grams
```

```
#=========================================#

t1 <- head(dt_tri[order(-Prob)], 10)
t1 <- t1[, trigram := paste(word1, word2, word3)][, .(trigram, Prob)]

t2 <- tail(dt_tri[order(-Prob)], 10)
t2 <- t2[, trigram := paste(word1, word2, word3)][, .(trigram, Prob)]

kable(list(t1, t2), "latex", booktabs = T,
      caption = "10-Most (Left) and 10-Least (Right) Likely Tri-Grams", align = "c") %>%
        kable_styling(latex_options = "hold_position")
```

```
#=========================================#

# Most and Least Likely Quad-Grams

#=========================================#

t1 <- head(dt_quad[order(-Prob)], 10)
t1 <- t1[, quadgram := paste(word1, word2, word3, word4)][, .(quadgram, Prob)]

t2 <- tail(dt_quad[order(-Prob)], 10)
t2 <- t2[, quadgram := paste(word1, word2, word3, word4)][, .(quadgram, Prob)]

kable(list(t1, t2), "latex", booktabs = T,
      caption = "10-Most (Left) and 10-Least (Right) Likely Quad-Grams", align = "c") %>%
        kable_styling(latex_options = "hold_position")
```

```
#=========================================#

# Most and Least Likely Quint-Grams

#=========================================#

t1 <- head(dt_quint[order(-Prob)], 10)
t1 <- t1[, quintgram := paste(word1, word2, word3, word4, word5)]
t1 <- t1[, .(quintgram, Prob)]

t2 <- tail(dt_quint[order(-Prob)], 10)
t2 <- t2[, quintgram := paste(word1, word2, word3, word4, word5)]
t2 <- t2[, .(quintgram, Prob)]


kable(list(t1, t2), "latex", booktabs = T,
      caption = "10-Most (Left) and 10-Least (Right) Likely Quint-Grams", align = "c") %>%
        kable_styling(latex_options = "hold_position")
```

```
#=========================================#

# Keep top-50 Uni-Grams

#=========================================#
```

```
dt_uni <- dt_uni[order(-Prob)][1:50]
```

```
#=========================================#

# Keep top-5 Bi-Grams for each Uni-Gram

#=========================================#


dt_bi <- dt_bi[, .SD[1:5], by = word1 ][!is.na(word2)]
```

```
#=========================================#

# Keep top-5 Tri-Grams for each Bi-Gram

#=========================================#


dt_tri <- dt_tri[, .SD[1:5], by = .(word1, word2) ][!is.na(word3)]
```

```
#=========================================#

# Keep top-5 Quad-Grams for each Tri-Gram

#=========================================#


dt_quad <- dt_quad[, .SD[1:5], by = .(word1, word2, word3) ][!is.na(word4)]
```

```
#=========================================#

# Keep top-5 Quint-Grams for each Quad-Gram

#=========================================#


dt_quint <- dt_quint[, .SD[1:5], by = .(word1, word2, word3, word4) ][!is.na(word5)]
```

```
#=========================================#

# top-5 Bi-Grams starting with "a"

#=========================================#

t <- dt_bi[word1=="a"]

kable(t, "latex", booktabs = T,
      caption = "Top-5 Bi-Grams starting with 'a'", align = "c") %>%
        kable_styling(latex_options = "hold_position")
```

```
#=========================================#

# top-5 Tri-Grams starting with "a lot"

#=========================================#
```

```
t <- dt_tri[word1=="a" & word2=="lot"]

kable(t, "latex", booktabs = T,
      caption = "Top-5 Tri-Grams starting with 'a lot'", align = "c") %>%
        kable_styling(latex_options = "hold_position")
```

```
#=========================================#

# top-5 Quad-Grams starting with "a lot of"

#=========================================#

t <- dt_quad[word1=="a" & word2=="lot" & word3=="of"]

kable(t, "latex", booktabs = T,
      caption = "Top-5 Tri-Grams starting with 'a lot of'", align = "c") %>%
        kable_styling(latex_options = "hold_position")
```

```
#=========================================#

# top-5 Quint-Grams starting with "a lot of people"

#=========================================#

t <- dt_quint[word1 =="a" & word2=="lot" & word3=="of" & word4=="people"]

kable(t, "latex", booktabs = T,
      caption = "Top-5 Quint-Grams starting with 'a lot of people'", align = "c") %>%
        kable_styling(latex_options = "hold_position")
```