Basic Natural Language Processing (NLP) in R

Mykyta Zharov

2/11/2020

Overview

This is a milestone report for the Data Science Specialization Capstone Project on Coursera. The main goal of the project is to build a machine learning algorithm in R which will be able to predict the next word the user intends to type and build a user-friendly interface around it (web-application). The project is motivated by SwiftKey virtual keyboard app.

The are three .txt files in English language provided for the task:

- collection of tweets from Twitter (167 mb)
- collection of blog entries (210 mb)
- collection of news items (205 mb)

The goal of the report is to present the following steps:

- Data reading
- Exploratory data analysis (EDA)
- Data preprocessing and cleaning
- Introducing the idea of a predictive algorithm based on n-Grams

Data reading

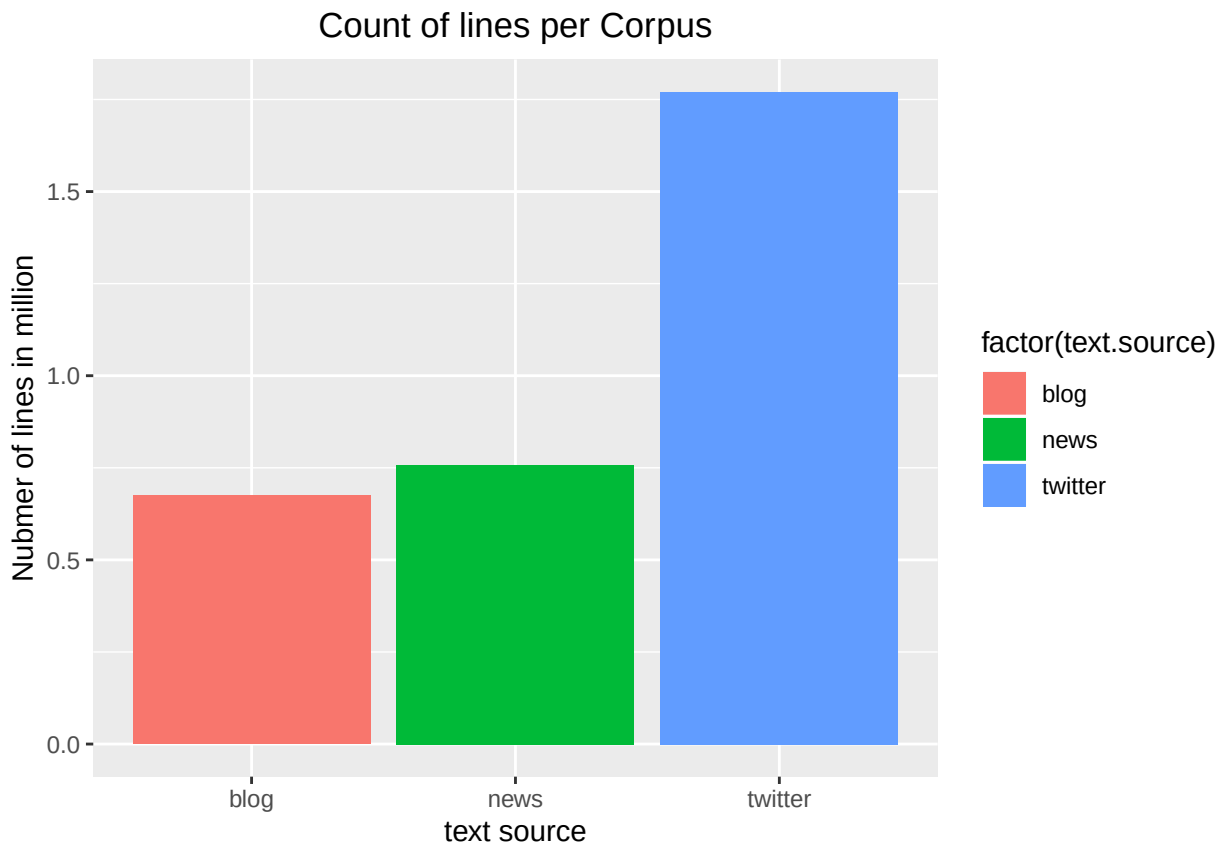
We start by loading the .txt files and, splitting into test and train datasets and saving them into single lists.

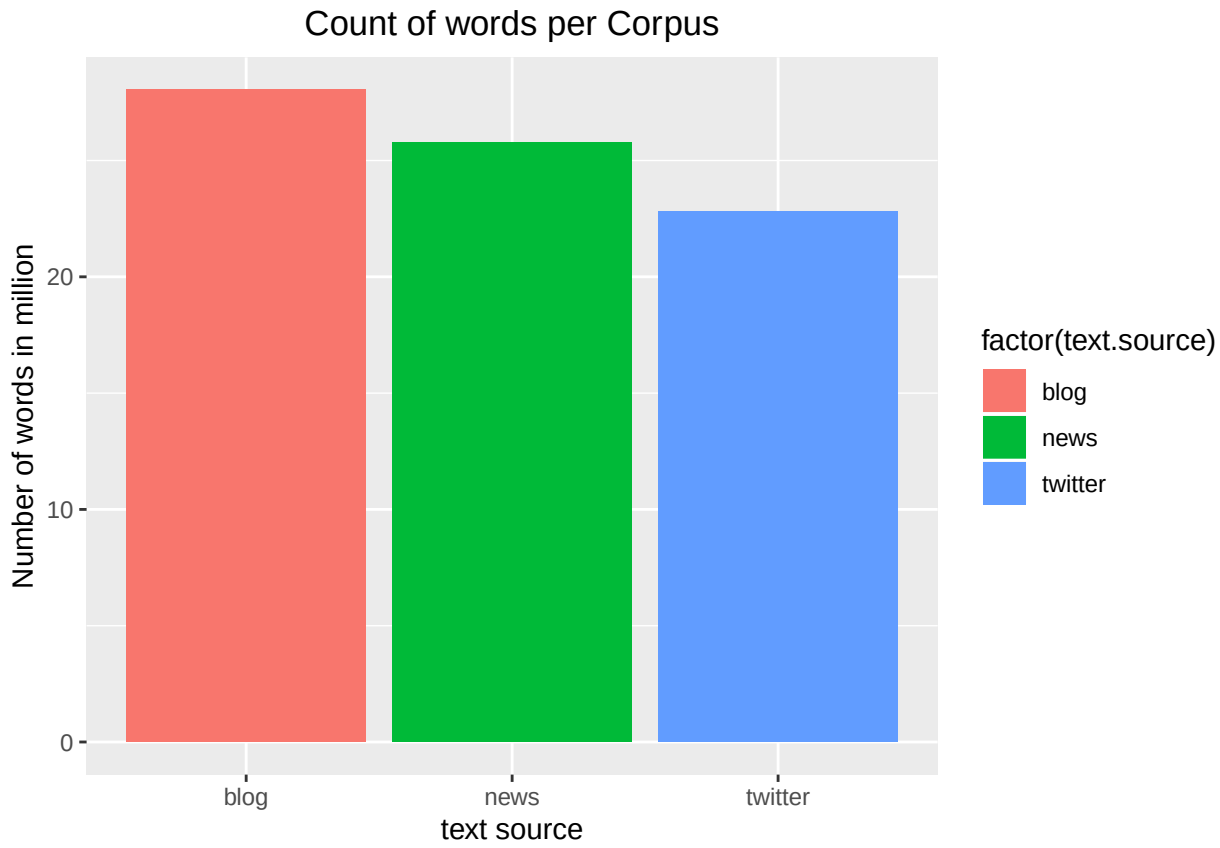
```
#set seed
set.seed(123)

#read files
blogs_text <- readLines("/Users/nikita/Documents/JH_ML_coursera/projects/capstone/final/en_US/en_US.blo
twitter_text <- readLines("/Users/nikita/Documents/JH_ML_coursera/projects/capstone/final/en_US/en_US.t
news_text <- readLines("/Users/nikita/Documents/JH_ML_coursera/projects/capstone/final/en_US/en_US.news

#split into test and train
smp_size_blogs <- floor(0.75 * length(blogs_text))
smp_size_twitter <- floor(0.75 * length(twitter_text))
smp_size_news <- floor(0.75 * length(news_text))

train_ind_blogs <- sample(seq_len(length(blogs_text)), size = smp_size_blogs)
train_ind_twitter <- sample(seq_len(length(twitter_text)), size = smp_size_twitter)
train_ind_news <- sample(seq_len(length(news_text)), size = smp_size_news)

train_blogs <- blogs_text[train_ind_blogs]
test_blogs <- blogs_text[-train_ind_blogs]
train_twitter <- twitter_text[train_ind_twitter]
test_twitter <- twitter_text[-train_ind_twitter]
train_news <- news_text[train_ind_news]
test_news <- news_text[-train_ind_news]
```

```
text_data <- list(blog = train_blogs, twitter = train_twitter, news = train_news)
text_data_test <- list(blog = test_blogs, twitter = test_twitter, news = test_news)

#save the test data set for further testing purposes later
saveRDS(text_data_test, file="text_data_test.rds")
```

Let us have a look at the number of lines and number of words we have in each text file in our training set.

```
# function to count number of words in a list
count_words <- function(list) { sum(stringr::str_count(list, "\\S+")) }

counts_df <- data.frame(text.source = c("blog", "twitter", "news"), line_count = NA, word_count = NA)
# get counts of lines and words for each text file
counts_df$line_count <- sapply(text_data, length)
counts_df$word_count <- sapply(text_data, count_words)

# plot counts for data sets
line_count_plot <- ggplot(counts_df, aes(x = factor(text.source), y = line_count/1e+06, fill=factor(tex
line_count_plot <- line_count_plot + geom_bar(stat = "identity") +
  labs(y = "Nubmer of lines in million", x = "text source", title = "Count of lines per Corpus")+
  theme(plot.title = element_text(hjust = 0.5))

word_count_plot <- ggplot(counts_df, aes(x = factor(text.source), y = word_count/1e+06, fill=factor(tex
word_count_plot <- word_count_plot + geom_bar(stat = "identity") +
  labs(y = "Number of words in million", x = "text source", title = "Count of words per Corpus")+
  theme(plot.title = element_text(hjust = 0.5))
```





From the plots above we see that twitter data has the most lines, but at the same time has less words than other datasets.

Exploratory data analysis

Since the data sets are fairly large, we are going to use only parts of them to build the predictive algorithm. We are going to create a separate sub-samples datasets by looking at a random subsets of the original train data. Let us think about this sub-samples as a of representative samples of a population quantity. We will use a binomial distribution with probability of success 3% to decide whether we sample a line of text from the original train dataset or not. This means that we are taking approximately 3% of lines from the original train data sets.

```
set.seed(1234)

percent <- 0.01
#create random numbers from binomial distribution for each text file
randoms <- lapply(text_data, function (x) rbinom(x, 1, percent))

random_selection <- list(blog = NA, twitter = NA, news = NA)
#perform random selection according to random numbers from the binomial distribution
for (i in 1:length(text_data)) {
  random_selection[[i]] <- text_data[[i]][randoms[[i]] == 1]
}

# calculate lines and words counts for obtained samples
samples <- data.frame(text.source = c("blog", "twitter", "news"),
```

```
                line_count = NA, word_count = NA)
samples$line_count <- sapply(random_selection, length)
samples$word_count <- sapply(random_selection, count_words)
samples
```

```
##   text.source line_count word_count
## 1        blog       6909     287513
## 2     twitter      17752     228306
## 3        news       7505     255574
```

From above output we see that all dataset have number of words ranging from 22 to 28 thousend.

Data preprocessing and cleaning

To further use the obtained samples to build a predictive algorithm we need to perform some data cleaning and preprocessing steps to increase the quality of the data and further prediction. We are going to use the "tm" (text mining) package to build a text corpus for each of the samples and perform preprocessing steps. More information about the text corpus and "tm" package can be found here: https://cran.r-project.org/web/packages/tm/vignettes/tm.pdf

```
#import tm package
library(tm)
```

```
## Loading required package: NLP
```

```
##
## Attaching package: 'NLP'
```

```
## The following object is masked from 'package:ggplot2':
##
##     annotate
```

```
#define functions for cleaning
deleteURL <- function(x) gsub("http:[[:alnum:]]*", "", x)
deleteEmail <- function(x) gsub("^([a-z0-9_\\.-]+)@([\\da-z\\.-]+)\\.([a-z\\.]{2,6})$", "", x)
deleteApostroph1<- function(x) gsub("'ll", " will", x)
deleteApostroph2<- function(x) gsub("'d", " would", x)
deleteApostroph3<- function(x) gsub("can't", "cannot", x)
deleteApostroph4<- function(x) gsub("n't", " not", x)
deleteApostroph5<- function(x) gsub("'re", " are", x)
deleteApostroph6<- function(x) gsub("'m", " am", x)
deleteApostroph7<- function(x) gsub("n'", " and", x)
deleteApostroph8<- function(x) gsub("dont", "do not", x)
# create VCorpus
text_corpus <- VCorpus(VectorSource(random_selection))
# delete spaces in the beginning and end of the line
text_corpus <- tm_map(text_corpus, stripWhitespace)
# change text to lowercase
text_corpus <- tm_map(text_corpus, content_transformer(tolower))
#delete emails
text_corpus <-tm_map(text_corpus, content_transformer(deleteEmail))
# delete numbers
text_corpus <- tm_map(text_corpus, removeNumbers)
# delete URL's
text_corpus <- tm_map(text_corpus, content_transformer(deleteURL))
#remove profanity
```

```
profanitySource <- VectorSource(readLines("profanity.txt"))
text_corpus = tm_map(text_corpus, removeWords, profanitySource)
# delete puctuation marks
text_corpus <- tm_map(text_corpus, removePunctuation)
#delete apostrophes
text_corpus <- tm_map(text_corpus, content_transformer(deleteApostroph1))
text_corpus <- tm_map(text_corpus, content_transformer(deleteApostroph2))
text_corpus <- tm_map(text_corpus, content_transformer(deleteApostroph3))
text_corpus <- tm_map(text_corpus, content_transformer(deleteApostroph4))
text_corpus <- tm_map(text_corpus, content_transformer(deleteApostroph5))
text_corpus <- tm_map(text_corpus, content_transformer(deleteApostroph6))
text_corpus <- tm_map(text_corpus, content_transformer(deleteApostroph7))
text_corpus <- tm_map(text_corpus, content_transformer(deleteApostroph8))
```

For the the twitter sample we perform additional cleaning steps.

```
# delete hashtags and nicknames
deleteHashTags <- function(x) gsub("#\\S+", "", x)
deleteNicknames <- function(x) gsub("@\\S+", "", x)
deleteTwitterLanguage1<- function(x) gsub("RT", "", x)
deleteTwitterLanguage2<- function(x) gsub("PM", "", x)
deleteTwitterLanguage3<- function(x) gsub("rt", "", x)
deleteTwitterLanguage4<- function(x) gsub("pm", "", x)
text_corpus["twitter"] <- tm_map(text_corpus["twitter"],
                         content_transformer(deleteHashTags))
```

```
## Warning in text_corpus["twitter"] <- tm_map(text_corpus["twitter"],
## content_transformer(deleteHashTags)): number of items to replace is not a
## multiple of replacement length
```

```
text_corpus["twitter"] <- tm_map(text_corpus["twitter"],
                         content_transformer(deleteNicknames))
```

```
## Warning in text_corpus["twitter"] <- tm_map(text_corpus["twitter"],
## content_transformer(deleteNicknames)): number of items to replace is not a
## multiple of replacement length
```

```
text_corpus["twitter"] <- tm_map(text_corpus["twitter"],
                         content_transformer(deleteTwitterLanguage1))
```

```
## Warning in text_corpus["twitter"] <- tm_map(text_corpus["twitter"],
## content_transformer(deleteTwitterLanguage1)): number of items to replace is
## not a multiple of replacement length
```

```
text_corpus["twitter"] <- tm_map(text_corpus["twitter"],
                         content_transformer(deleteTwitterLanguage2))
```

```
## Warning in text_corpus["twitter"] <- tm_map(text_corpus["twitter"],
## content_transformer(deleteTwitterLanguage2)): number of items to replace is
## not a multiple of replacement length
```

```
text_corpus["twitter"] <- tm_map(text_corpus["twitter"],
                         content_transformer(deleteTwitterLanguage3))
```

```
## Warning in text_corpus["twitter"] <- tm_map(text_corpus["twitter"],
## content_transformer(deleteTwitterLanguage3)): number of items to replace is
## not a multiple of replacement length
```

```
text_corpus["twitter"] <- tm_map(text_corpus["twitter"],
                          content_transformer(deleteTwitterLanguage4))
```

```
## Warning in text_corpus["twitter"] <- tm_map(text_corpus["twitter"],
## content_transformer(deleteTwitterLanguage4)): number of items to replace is
## not a multiple of replacement length
```

We also delete all non ascii characters. In this way we remove foreign languages.

```
# delete non asciii characters
delete_non_ascii <- function(x) iconv(x, "latin1", "ASCII", sub="")
text_corpus <- tm_map(text_corpus, content_transformer(delete_non_ascii))
```

Now the corpus is cleaned and is ready for building the n-gram model.

Idea of a predictive algorithm based on n-Grams

As a ferst step to buid a predictive algorithm based on n-grams (https://en.wikipedia.org/wiki/N-gram) we are going to calculate 1-grams, the frequencies of separate words in a corpus. It can be easily done by building a term-document matrix (tdm). A term-document matrix is a mathematical matrix that describes the frequency of terms that occur in a collection of documents. In a term-document matrix, rows correspond to documents in the collection and columns correspond to terms.

```
#create 1-gram tokenizer
uniGramTokenizer <- function(x) {
      unlist(lapply(ngrams(words(x), 1), paste, collapse = " "), use.names = FALSE)
}

# compute the tdm matrix for 3 samples together and
tdm <- TermDocumentMatrix(text_corpus, control = list(tokenize = uniGramTokenizer))
tdm
```

```
## <<TermDocumentMatrix (terms: 48084, documents: 3)>>
## Non-/sparse entries: 72510/71742
## Sparsity           : 50%
## Maximal term length: 88
## Weighting          : term frequency (tf)
```

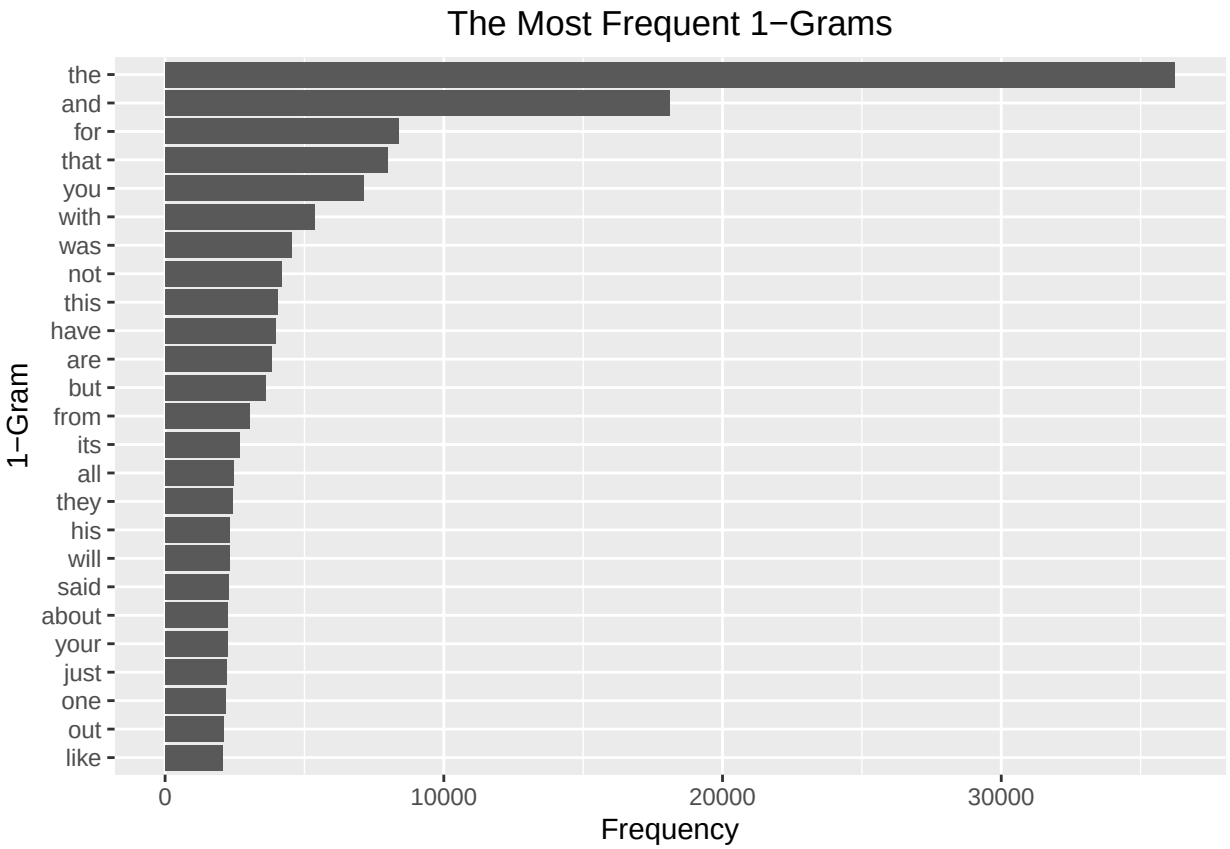
From the above output we see that we have 50% sparsity, which means that 50% of the cells in the tdm matrix are zeros. We want to bring more efficency to the model and therefore we can delete sparse words with threshold 10%.

```
#tdm <- removeSparseTerms(tdm, 0.1)
```

We can now see which words have the highest frequency in all of the samples. Let us calculate the top 30 words according to frequency.

```
# calculate word frequencies
freqTerms <- findFreqTerms(tdm, lowfreq = 2000)
wordFrequency <- rowSums(as.matrix(tdm[freqTerms,]))
wordFrequency <- data.frame(unigram=names(wordFrequency), frequency=wordFrequency)

# reorder by descreasing frequency
wordFrequency <- wordFrequency[order(-wordFrequency$frequency),]

g<-ggplot(head(wordFrequency, 30), aes(x=reorder(unigram, frequency), y=frequency)) +
  geom_bar(stat = "identity") +  coord_flip() +
```

```
  theme(legend.title=element_blank()) +
  xlab("1-Gram") + ylab("Frequency") +
  labs(title = "The Most Frequent 1-Grams")+
  theme(plot.title = element_text(hjust = 0.5))
g
```



Now we will also have a look at 2-Grams and 3-Grams.

```
# create tokenizers
biGramTokenizer <- function(x) {
      unlist(lapply(ngrams(words(x), 2), paste, collapse = " "), use.names = FALSE)
}
triGramTokenizer <- function(x) {
      unlist(lapply(ngrams(words(x), 3), paste, collapse = " "), use.names = FALSE)
}
fourGramTokenizer <- function(x) {
      unlist(lapply(ngrams(words(x), 4), paste, collapse = " "), use.names = FALSE)
}
# create 2- and 3- and 4-term matrices
biGram_tdm <- TermDocumentMatrix(text_corpus, control = list(tokenize = biGramTokenizer))
biGram_tdm
```

```
## <<TermDocumentMatrix (terms: 374142, documents: 3)>>
## Non-/sparse entries: 435187/687239
## Sparsity           : 61%
## Maximal term length: 92
## Weighting          : term frequency (tf)
```

```
triGram_tdm <- TermDocumentMatrix(text_corpus, control = list(tokenize = triGramTokenizer ))
triGram_tdm
```

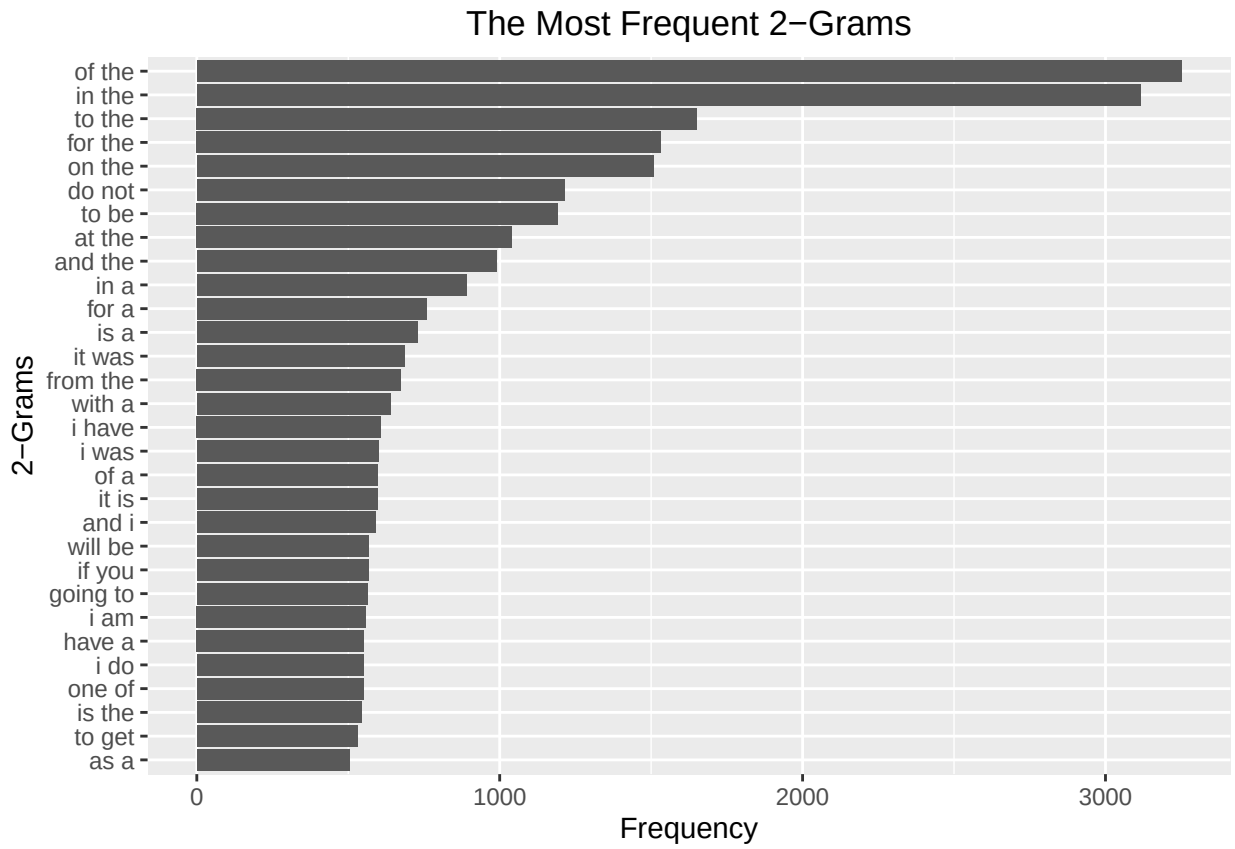
```
## <<TermDocumentMatrix (terms: 651936, documents: 3)>>
## Non-/sparse entries: 681033/1274775
## Sparsity           : 65%
## Maximal term length: 96
## Weighting          : term frequency (tf)
```

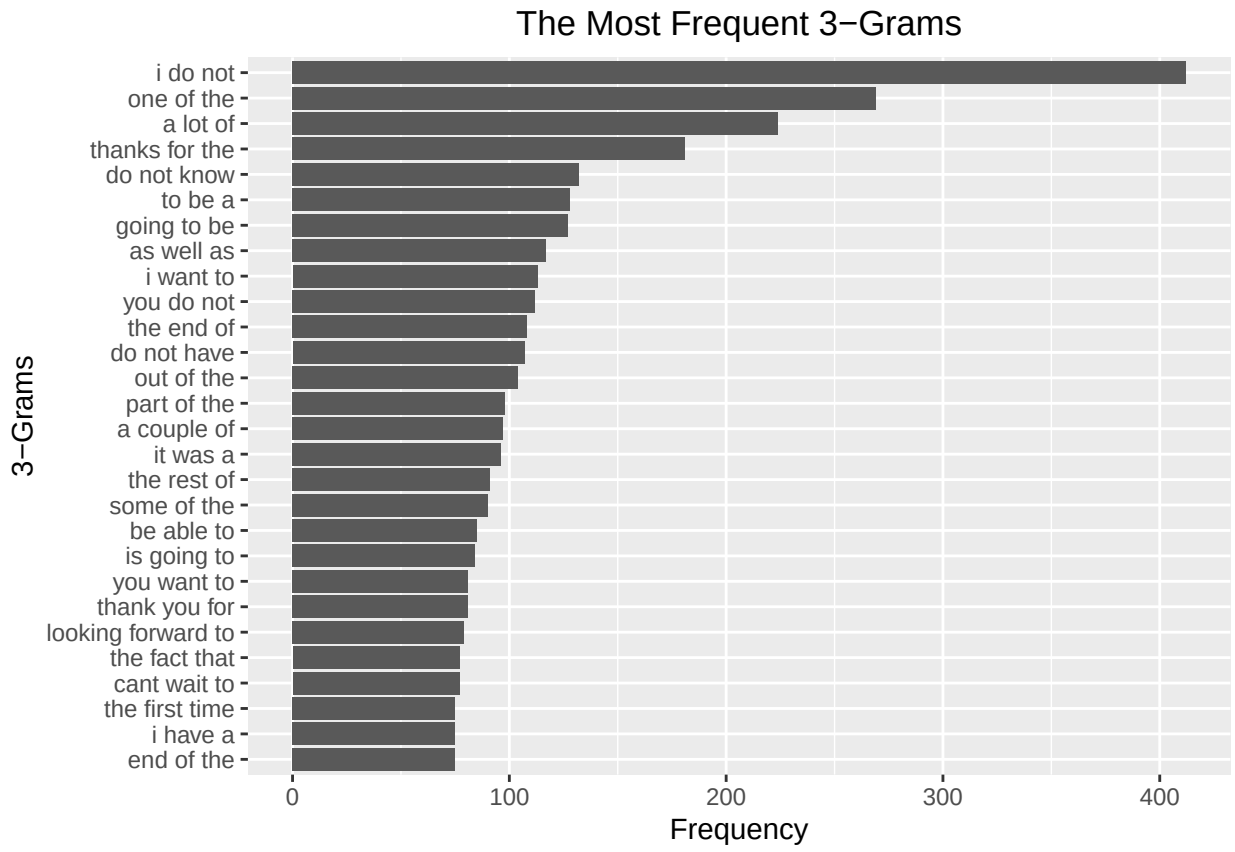
```
fourGram_tdm <- TermDocumentMatrix(text_corpus, control = list(tokenize = fourGramTokenizer ))
fourGram_tdm
```

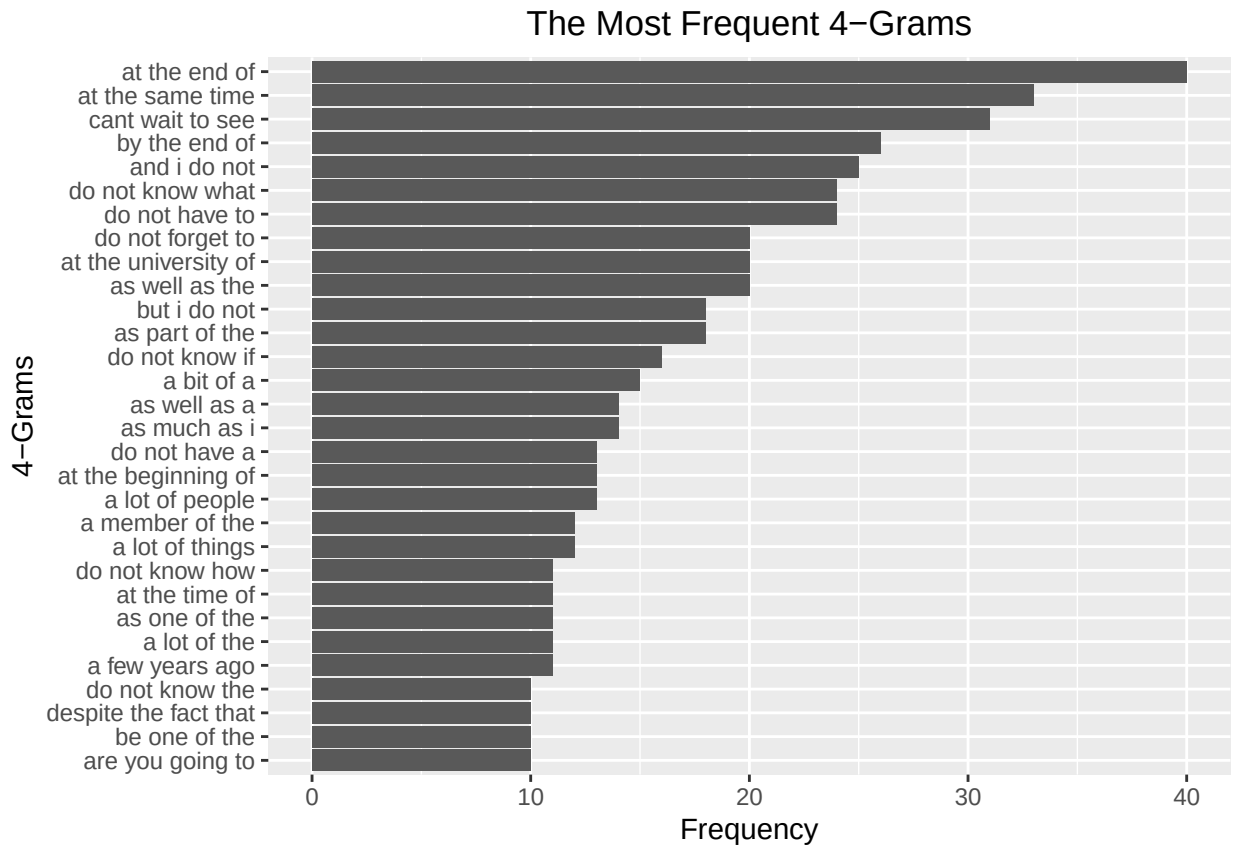
```
## <<TermDocumentMatrix (terms: 734547, documents: 3)>>
## Non-/sparse entries: 740465/1463176
## Sparsity           : 66%
## Maximal term length: 104
## Weighting          : term frequency (tf)
```

```
# delete sparse terms, as for tdm above
#biGram_tdm <- removeSparseTerms(biGram_tdm, 0.001)
#triGram_tdm <- removeSparseTerms(triGram_tdm, 0.001)
#fourGram_tdm <- removeSparseTerms(fourGram_tdm, 0.001  )

freqTerms <- findFreqTerms(biGram_tdm, lowfreq = 500)
wordFrequency_bi <- rowSums(as.matrix(biGram_tdm[freqTerms,]))
wordFrequency_bi <- data.frame(bigram=names(wordFrequency_bi), frequency=wordFrequency_bi)

g2 <- ggplot(head(wordFrequency_bi, 30), aes(x=reorder(bigram, frequency), y=frequency)) +
  geom_bar(stat = "identity") +  coord_flip() +
  theme(legend.title=element_blank()) +
  xlab("2-Grams") + ylab("Frequency") +
  labs(title = "The Most Frequent 2-Grams")+
  theme(plot.title = element_text(hjust = 0.5))


freqTerms <- findFreqTerms(triGram_tdm, lowfreq = 75)
wordFrequency_tri <- rowSums(as.matrix(triGram_tdm[freqTerms,]))
wordFrequency_tri <- data.frame(trigram=names(wordFrequency_tri), frequency=wordFrequency_tri)

g3 <- ggplot(head(wordFrequency_tri, 30), aes(x=reorder(trigram, frequency), y=frequency)) +
  geom_bar(stat = "identity") +  coord_flip() +
  theme(legend.title=element_blank()) +
  xlab("3-Grams") + ylab("Frequency") +
  labs(title = "The Most Frequent 3-Grams")+
  theme(plot.title = element_text(hjust = 0.5))

freqTerms <- findFreqTerms(fourGram_tdm, lowfreq = 10)
wordFrequency_four <- rowSums(as.matrix(fourGram_tdm[freqTerms,]))
wordFrequency_four <- data.frame(fourgram=names(wordFrequency_four), frequency=wordFrequency_four)

g4 <- ggplot(head(wordFrequency_four, 30), aes(x=reorder(fourgram, frequency), y=frequency)) +
  geom_bar(stat = "identity") +  coord_flip() +
  theme(legend.title=element_blank()) +
  xlab("4-Grams") + ylab("Frequency") +
  labs(title = "The Most Frequent 4-Grams")+
```

```
theme(plot.title = element_text(hjust = 0.5))
```




The Most Frequent 3−Grams
3−Grams
i do not
one of the
a lot of
thanks for the
do not know
to be a
going to be
as well as
i want to
you do not
the end of
do not have
out of the
part of the
a couple of
it was a
the rest of
some of the
be able to
is going to
you want to
thank you for
looking forward to
the fact that
cant wait to
the first time
i have a
end of the
0
100
200
300
400
Frequency


The Most Frequent 4−Grams
4−Grams
at the end of
at the same time
cant wait to see
by the end of
and i do not
do not know what
do not have to
do not forget to
at the university of
as well as the
but i do not
as part of the
do not know if
a bit of a
as well as a
as much as i
do not have a
at the beginning of
a lot of people
a member of the
a lot of things
do not know how
at the time of
as one of the
a lot of the
a few years ago
do not know the
despite the fact that
be one of the
are you going to
0
10
20
30
40
Frequency

At this stage we can save our tdm matrices as .rds files not to compute them again every time we want to run the algotithm in the future.

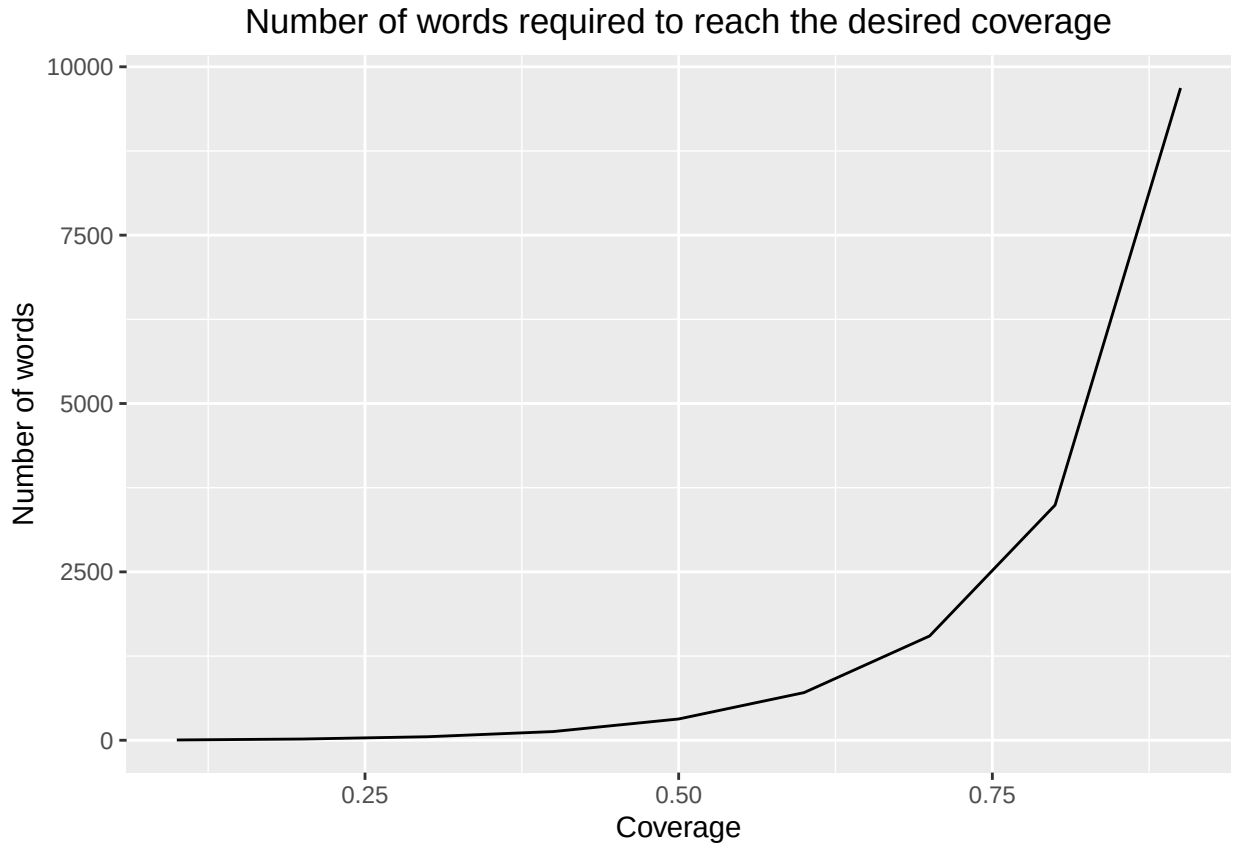
```
saveRDS(tdm, file="tdm.rds")
saveRDS(biGram_tdm, file="bigram.rds")
saveRDS(triGram_tdm, file="trigram.rds")
saveRDS(fourGram_tdm, file="fourgram.rds")
```

It is interesting to answer the following question: How many unique words do you need in a frequency sorted dictionary to cover 50% of all word instances in the language? To answer this question we can use the following function.

```
word_coverage<-function(x,word_cover) {  #x is the 1-gram output sorted by frequency, y is the percent
  nwords<-0
  coverage<-word_cover*sum(x$frequency)
  for (i in 1:nrow(x)) {
    if (nwords >= coverage) {return (i)}
    nwords<-nwords+x$frequency[i]
  }}
```

Let us calculate the answer to the question from above.

```
x <- seq(0.1, 0.9, by = 0.1)
y <- c()

freqTerms <- findFreqTerms(tdm)
wordFrequency <- rowSums(as.matrix(tdm[freqTerms,]))
wordFrequency <- data.frame(unigram=names(wordFrequency), frequency=wordFrequency)
wordFrequency <- wordFrequency[order(-wordFrequency$frequency),]

for(i in x)
{
  y[i*10] <- word_coverage(wordFrequency, i)
}

qplot(x, y, geom = c("line"), xlab="Coverage", ylab="Number of words", main="Number of words required t
  theme(plot.title = element_text(hjust = 0.5))
```



The plot from above clearly shows the exponential relationship between number of words and coverage. Using this fact we can consider setting the desired coverage level around 90%-95% while building the algorithm and hence significantly reducing the number of words needed.

Let us also have a quick look at how many bytes our n-Gramm matrices use in memory.

```
utils:::format.object_size(object.size(tdm), "auto")
```

```
## [1] "4.2 Mb"
```

```
utils:::format.object_size(object.size(biGram_tdm), "auto")
```

```
## [1] "32.8 Mb"
```

```
utils:::format.object_size(object.size(triGram_tdm), "auto")
```

```
## [1] "60.1 Mb"
```

```
utils:::format.object_size(object.size(fourGram_tdm), "auto")
```

```
## [1] "72.1 Mb"
```

We are interested in building an app that can work quickly, so we might have to improve the model size in the future. But taking 1% of the training data gives us a relatively small size.

Further we can use the obtained matrices to develop an n-Gram predictive algorithm. It can be implemented in the following way:

- Split 2-grams into 1-Gram/1-Gram pairs
- Split 3-grams into 2-Gram/1-Gram pairs
- Split 4-grams into 3-Gram/1-Gram pairs
- Store only the 3 most frequent occurences of the pairs.

We can then use the obtained information to give the 3 predictions for the user as follows:

- If the supplied text is greater than 3 words, take the last three words of the text and search the 3-Gram/1-Gram pairs.
- If the supplied text is 3 words, take the three words and search the 2-Gram/1-Gram pairs.
- If the supplied text is 2 words, search for that word in the 2-Gram/1-Gram pairs.
- If the supplied text is 1 word, search for that word in the 1-Gram/1-Gram pairs.

Further improvements

Above we have seen the basic idea how we can implement a simple n-Gram model. Such an implementation still has it's drawbacks. Some problems arise from the following questions:

- What are we going to do if we recive an unseen word?
- Is our model fast and accurate enough to run on the mobile device?
- How can we reduce the size of the model?

Further we are going to investigate the above questions and improve the algorithm respectively. Further we are going to build an 4-gram prediction algorithm, using the stupid backoff algorithm. This approach will solve the issue with the unseen n-grams. More information about the stupid backoff algorithm can be found here: https://www.aclweb.org/anthology/D07-1090.pdf. The accuracy and speed of the model can be then tested on the testing data set (4 grams from the test data set).

Below you can find additional code to build n-grams for the test data set. It can be used for further testing purposes.

```
#apply the same steps for the test dataset and save as .rds files

#take 10% of the original test dataset
# percent <- 0.1
# #create random numbers from binomial distribution for each text file
# randoms <- lapply(text_data_test, function (x) rbinom(x, 1, percent))
#
# random_selection <- list(blog = NA, twitter = NA, news = NA)
# #perform random selection according to random numbers from the binomial distribution
# for (i in 1:length(text_data_test)) {
#   random_selection[[i]] <- text_data_test[[i]][randoms[[i]] == 1]
# }
#
# # calculate lines and words counts for obtained samples
# samples <- data.frame(text.source = c("blog", "twitter", "news"),
#                   line_count = NA, word_count = NA)
# samples$line_count <- sapply(random_selection, length)
# samples$word_count <- sapply(random_selection, count_words)
# samples
#
# #perform data cleaning
#
# # create VCorpus
# text_corpus_test <- VCorpus(VectorSource(random_selection))
# # delete spaces in the beginning and end of the line
# text_corpus_test <- tm_map(text_corpus_test, stripWhitespace)
# # change text to lowercase
# text_corpus_test <- tm_map(text_corpus_test, content_transformer(tolower))
# # delete numbers
# text_corpus_test <- tm_map(text_corpus_test, removeNumbers)
```

```
# # delete URL's
# text_corpus_test <- tm_map(text_corpus_test, content_transformer(deleteURL))
# # delete puctuation marks
# text_corpus_test <- tm_map(text_corpus_test, removePunctuation)
#
# text_corpus_test["twitter"] <- tm_map(text_corpus_test["twitter"],
#                           content_transformer(deleteHashTags))
# text_corpus_test["twitter"] <- tm_map(text_corpus_test["twitter"],
#                           content_transformer(deleteNicknames))
# text_corpus_test <- tm_map(text_corpus_test, content_transformer(delete_non_ascii))
#
# tdm_test <- TermDocumentMatrix(text_corpus_test, control = list(tokenize = uniGramTokenizer))
#
# biGram_tdm_test <- TermDocumentMatrix(text_corpus_test, control = list(tokenize = biGramTokenizer))
#
# triGram_tdm_test  <- TermDocumentMatrix(text_corpus_test, control = list(tokenize = triGramTokenizer))
#
# fourGram_tdm_test  <- TermDocumentMatrix(text_corpus_test, control = list(tokenize = fourGramTokenize
#
# #save
# saveRDS(tdm_test, file="tdm_test.rds")
# saveRDS(biGram_tdm_test, file="bigram_test.rds")
# saveRDS(triGram_tdm_test, file="trigram_test.rds")
# saveRDS(fourGram_tdm_test, file="fourgram_test.rds")
```