Next word prediction algorithm development

Ayush Bhargava

January 7, 2018

Introduction

This report is intended to demonstrate the inital steps taken towards the overall project goal of creating an NLP (Natural Language Processing) next word predictive algorighm based on documents provided. This is part of the Capstone class in the Coursera Data Science specialization and was designed in cooperation with Johns Hopkins University and Swiftkey, a software company specializing in creating soft keyboard apps with integrated next word predictictive algorighms.

The provided data files were processed. Summary statistices such as file size, number of words, etc. were gathered to better understand the sixe and contents of the files. Punction, profanity, numbers and white space was removed. Stemming was not done as this is more useful for sentiment analysis. For next word predicion we wanted to build on the actual words used. A word cloud was built to get a quick visual que as to what the most common words in the sample are. We then built and ran a tokenizing funciton to find the most common words (unigrams), word pairs (bigrams), three word sets (trigram) and quadgrams (4 word sets). Graphs of the 15 most common of each are shown below. Next stupid backoff and markov chain model is built to predict the next word

Data Source

Data for this project is available at coursera data science specialization. The downloaded data file contains three text files in three languages (english, russian and finnish). For this project we will only use the english files. The text files are a collection of blog posts, news articles and twitter posts. As the files are very large, samples of the files will be used to build the corpus from which we will create the predicive algorithm.

Analysing Full english data-set

Loading Required packages

```
library(stringi)
library(ggplot2)
library(magrittr)
library(markdown)
library(knitr)
library(RWeka)
library(openNLP)
library(wordcloud)
library(tm)
library(NLP)
library(qdap)
library(RColorBrewer)
library(dplyr)
```

Loading data

```
setwd("C:/Users/ichbi/Desktop/Data science specialization/Capstone project/Dataset/final/en_US")
twitter <- readLines("en_US.twitter.txt", encoding="UTF-8")
news <- readLines("en_US.news.txt", encoding="UTF-8")
blogs <- readLines("en_US.blogs.txt", encoding="UTF-8")
```

Summary of files

```
#writing some helper functions
size <- function(x){file.info(x)$size/1024/1024}
lines <- function(x){length(x)}
char <- function(x){sum(stri_length(x)-stri_count_fixed(x," "))}
words<- function(x){sum(stri_count_words(x))}
#creating summary file with the help of helper functions
filesummary<- data.frame(source=c("twitter","blogs","news"),
          file_size=c(size("en_US.twitter.txt"),size("en_US.blogs.txt"),size("en_US.news.txt")),
          no_of_lines=c(lines(twitter),lines(blogs),lines(news)),
          no_of_characters=c(char(twitter),char(blogs),char(news)),
          no_of_words=c(words(twitter),words(blogs),words(news)))
```

```
kable(filesummary, caption="Data summary")
```

Table 1: Data summary

source	file_size	no_of_lines	no_of_characters	no_of_words
twitter	159.3641	2360148	134082634	30093369
blogs	200.4242	899288	170389662	37546246
news	196.2775	77259	13072698	2674536
As the dat	a size is to	o big, we will	sample the data fo	r further exploration

Sampling the data

we will take 5000 entries from each data set for further exploration.For our model we do not want to predict bad words therefore we will use a profanity word list available at https://github.com/shutterstock/List-of-Dirty-Naughty-Obscene-and-Otherwise-Bad-Words/blob/master/en and remove these words from our data set

```
#creating connections
con1 <- file("en_US.twitter.txt", "r")
con2<- file("en_US.blogs.txt", "r")
con3<-file("en_US.news.txt", "r")
#Reading data
twitter <- readLines(con1,5000)
blogs<- readLines(con2,5000)
news<- readLines(con3,5000)
#closing connections
close(con1)
close(con2)
close(con3)
#reading badwords
```

```
badwords<- readLines("en_US.profane.txt")
```

we will now combine the data from all three sources, so that our data represent a variety of writing styles and then we will use english language to detect sentences

```
data <- paste(twitter,blogs,news)
data <- sent_detect(data,language="en",model=NULL)
```

cleaning and data prep

```
#removing non alphabets
data <- gsub("[^a-zA-Z ]","",data)

#creating corpus
corpus<- VCorpus(VectorSource(data))

#removing bad words
badwords <- VectorSource(badwords)

#corpus cleaning
corpus <- tm_map(corpus,removeNumbers)
corpus<- tm_map(corpus,removePunctuation)
corpus<- tm_map(corpus,stripWhitespace)
corpus <- tm_map(corpus, content_transformer(tolower))
corpus <- tm_map(corpus, removeWords, badwords)
```

Now our data is clean and ready for further analysis. Next step is to creat a list of words in our data set and associated frequency, this process is done by bag of words method or more specifically creating N-grams (unigram, bigram etc) and there after term document matrix for each gram

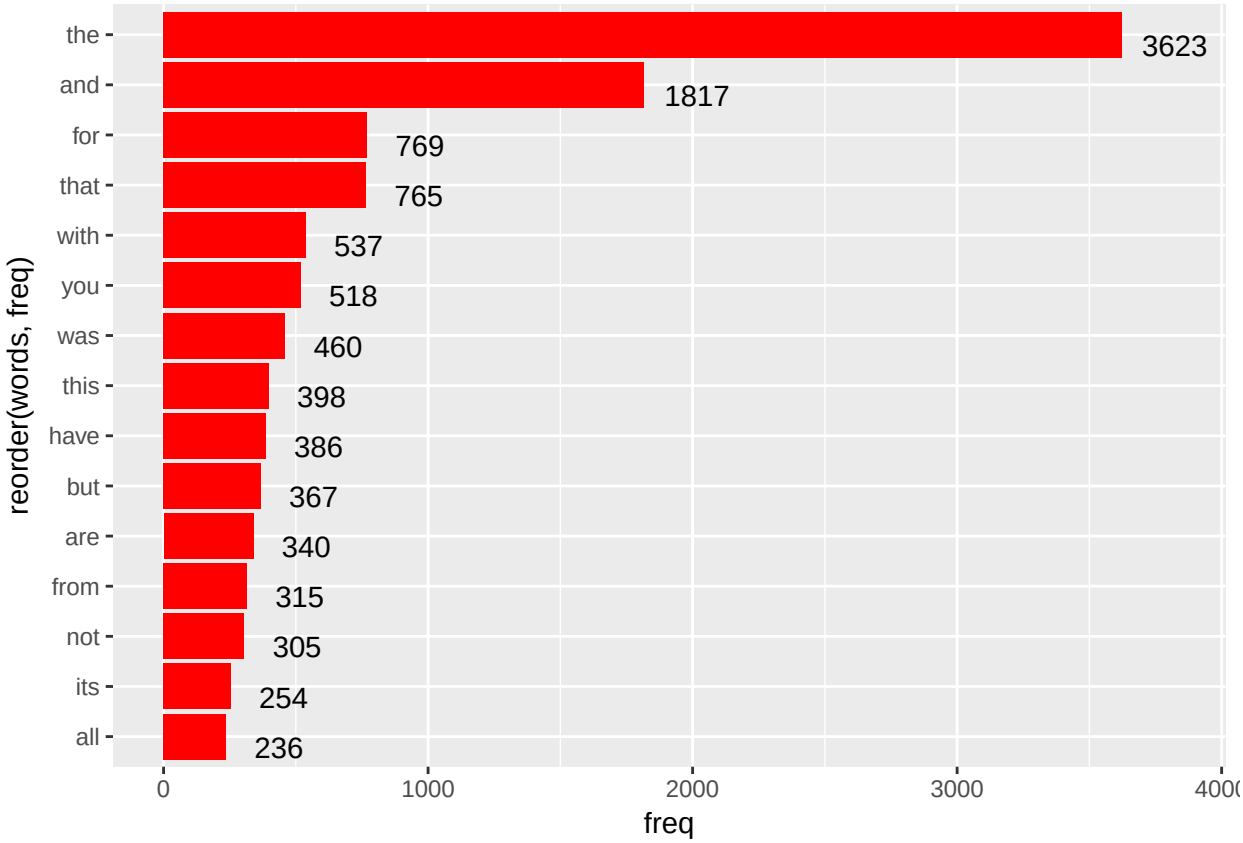
In this analysis, i have reffered unigram, bigram etc and onegram, twogram etc interchangeablly

Unigram

```
unitoken<- function(x){NGramTokenizer(x, Weka_control(min = 1, max = 1))}
tdm_for_onegram <- TermDocumentMatrix(corpus,control = list(tokenize = unitoken))
fm <- rowSums(as.matrix(tdm_for_onegram))
onegram <- data.frame(words=names(fm),freq=fm)
onegram <- onegram[order(onegram$freq, decreasing=TRUE),]
```

Now that we have our unigram, lets get the top 15 words and visualise them

```
top_onegram <- onegram[1:15,]
ggplot(top_onegram, aes(x=reorder(words,freq),y=freq))+geom_bar(stat="identity", fill="red")+ geom_text
```

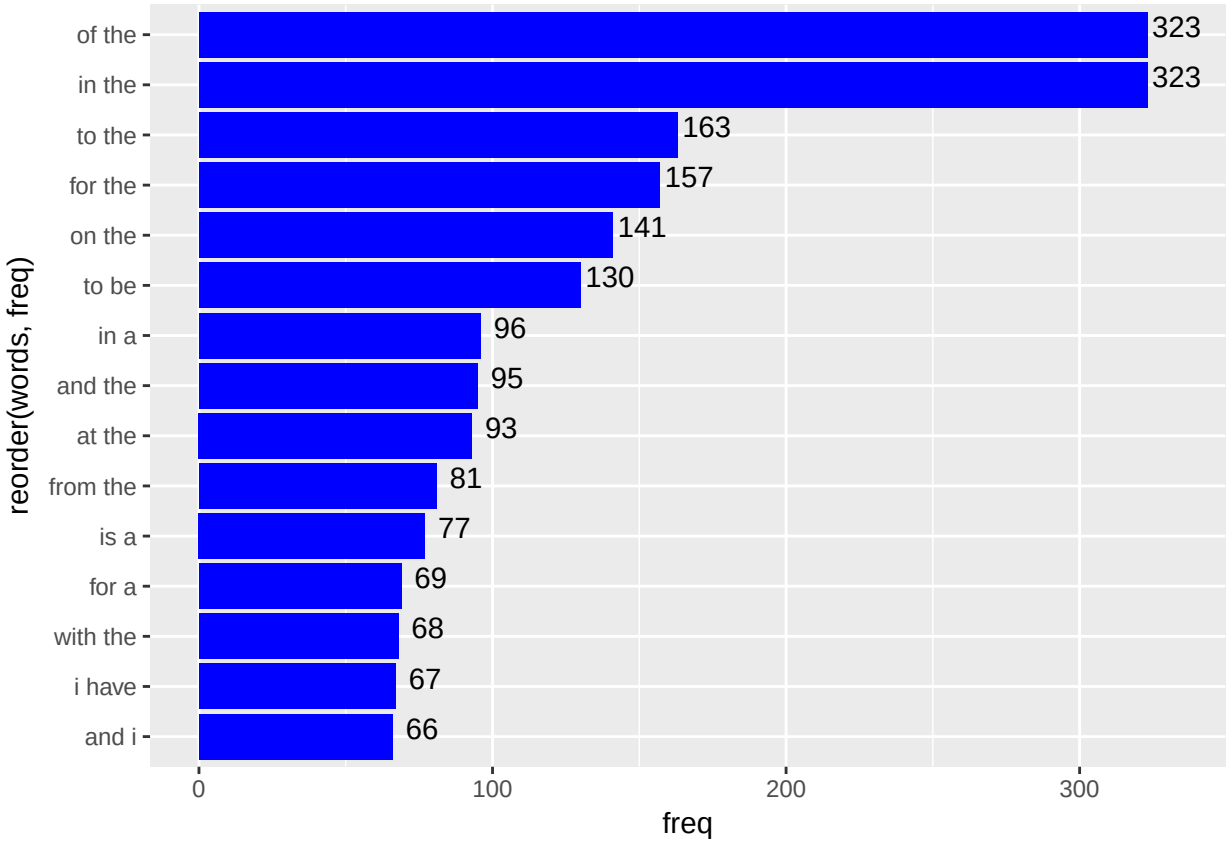



```
wordcloud(onegram$words,onegram$freq, max.words=50,random.order=FALSE,rot.per=0.35,colors=brewer.pal(8,
```

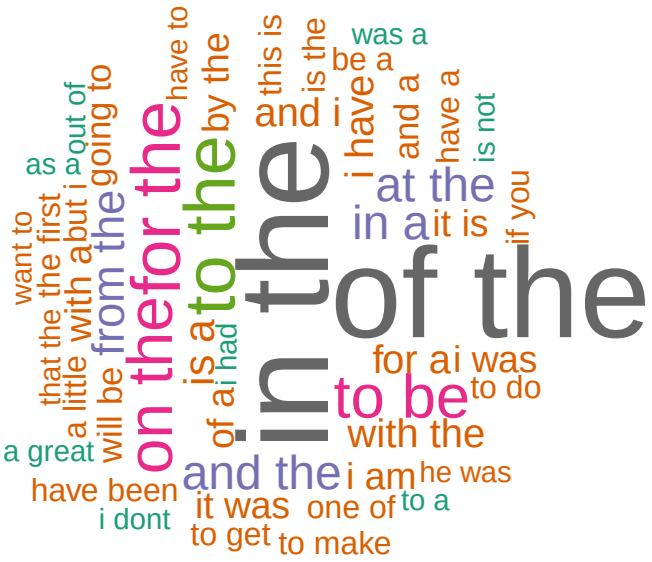
we can do similar plots for bigram, trigram and fourgram

Bigram

```
bitoken <- function(x){NGramTokenizer(x, Weka_control(min=2, max=2))}
tdm_for_twogram <- TermDocumentMatrix(corpus,control=list(tokenizer = bitoken))
fm <- rowSums(as.matrix(tdm_for_twogram))
twogram <- data.frame(words=names(fm),freq=fm)
twogram <- twogram[order(twogram$freq, decreasing=TRUE),]
top_twogram <- twogram[1:15,]
ggplot(top_twogram, aes(x=reorder(words,freq),y=freq))+geom_bar(stat="identity", fill="blue")+ geom_tex
```

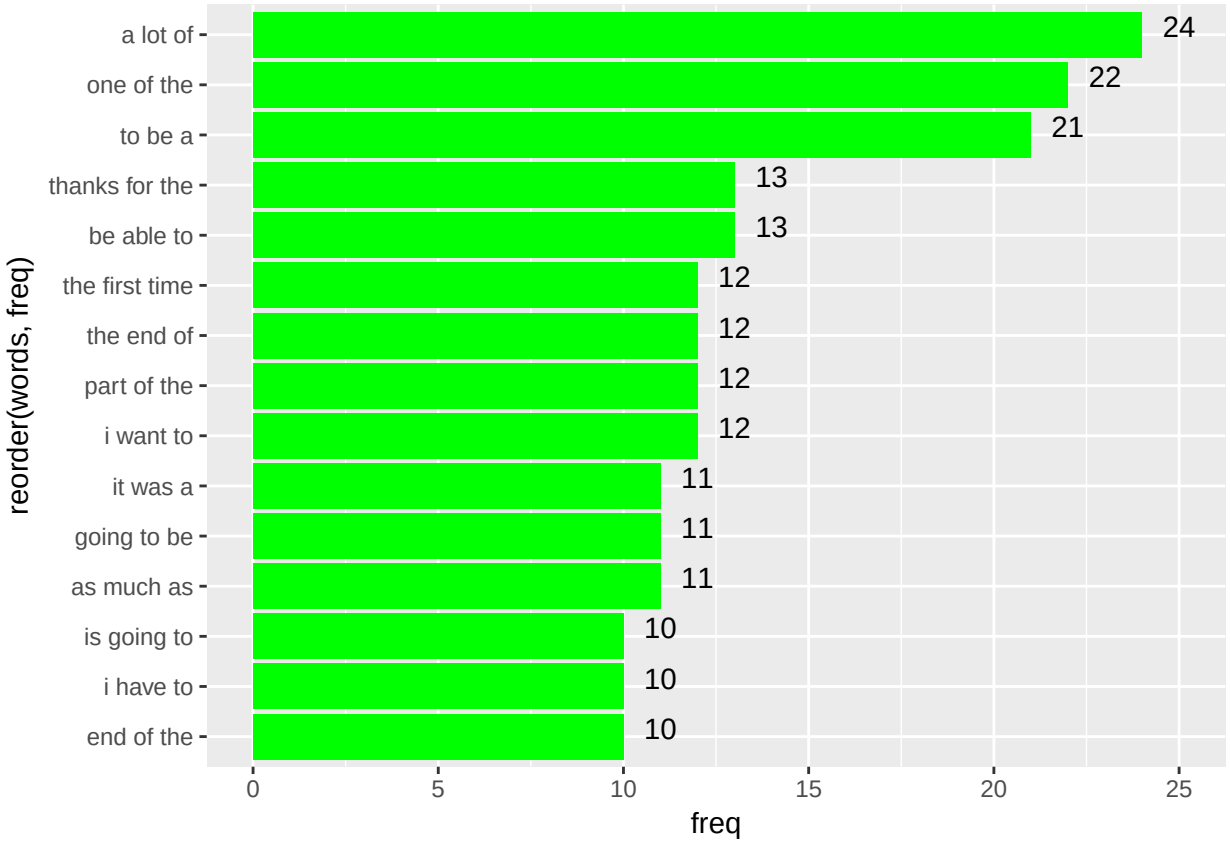



```
wordcloud(twogram$words,twogram$freq, max.words=50,random.order=FALSE,rot.per=0.35,colors=brewer.pal(8,
```

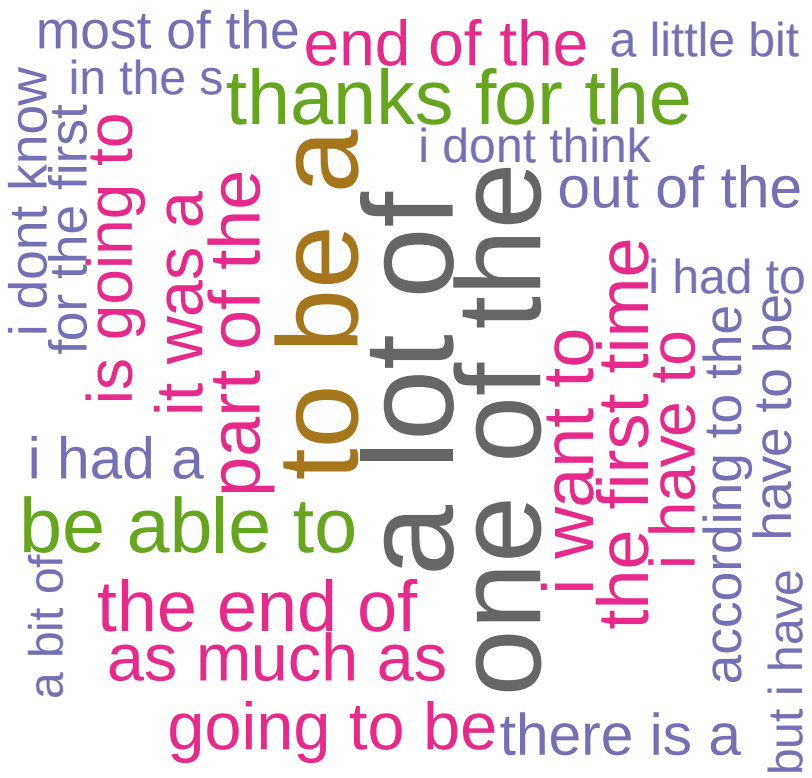


trigram

```
tritoken <- function(x){NGramTokenizer(x, Weka_control(min=3, max=3))}
tdm_for_threegram <- TermDocumentMatrix(corpus,control=list(tokenizer = tritoken))
fm <- rowSums(as.matrix(tdm_for_threegram))
threegram <- data.frame(words=names(fm),freq=fm)
threegram <- threegram[order(threegram$freq, decreasing=TRUE),]
top_threegram <- threegram[1:15,]
ggplot(top_threegram, aes(x=reorder(words,freq),y=freq))+geom_bar(stat="identity", fill="green")+ geom_
```

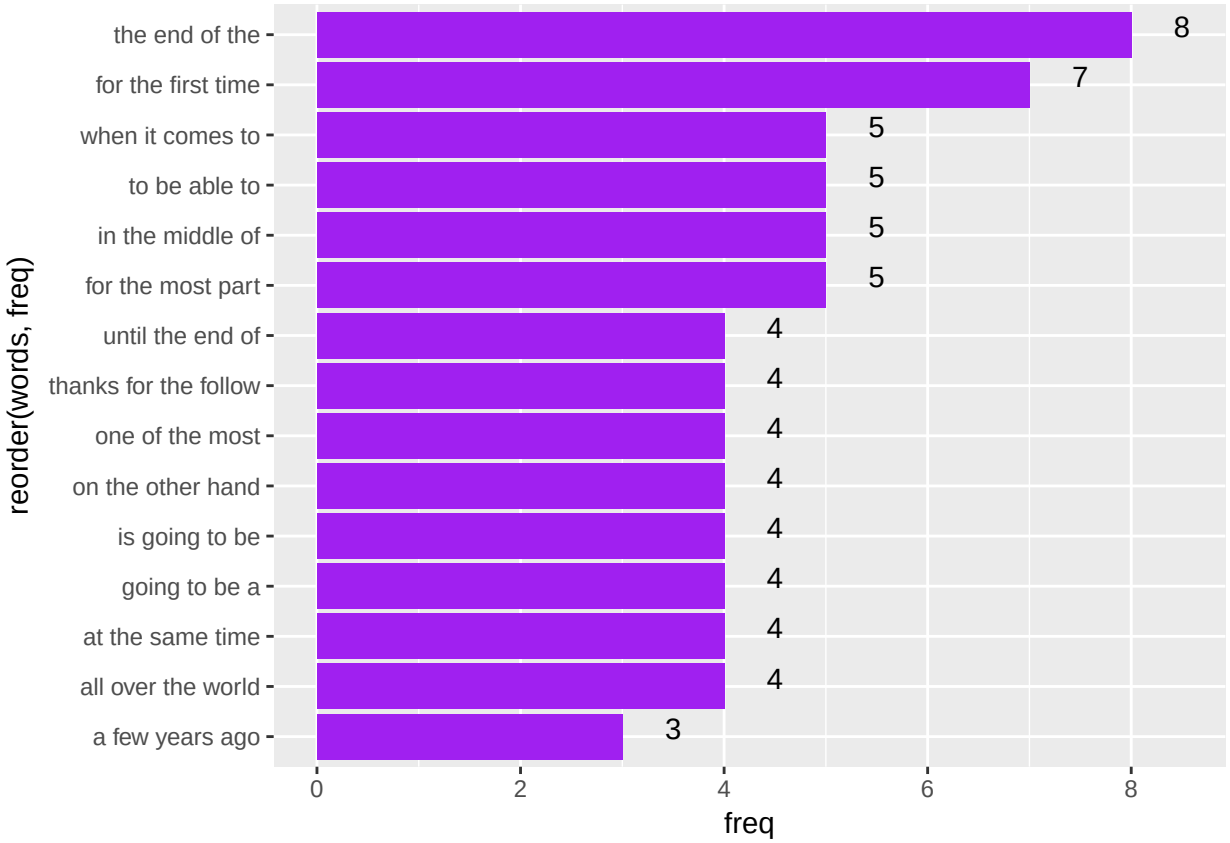



```
wordcloud(threegram$words,threegram$freq, max.words=50,random.order=FALSE,rot.per=0.50,colors=brewer.pal
```

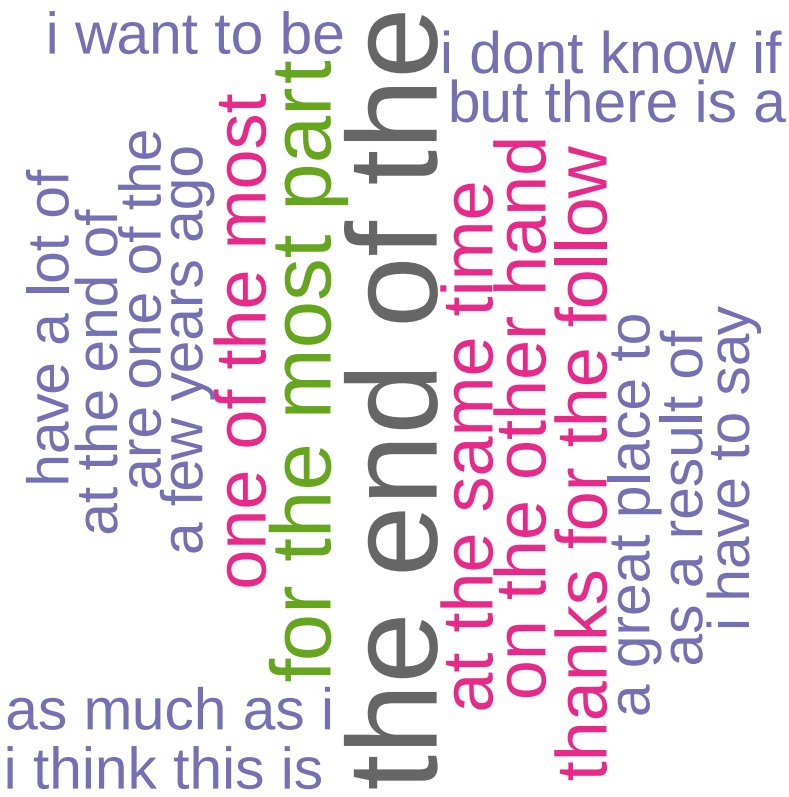


Fourgram

```
qudratoken <- function(x){NGramTokenizer(x, Weka_control(min=4, max=4))}
tdm_for_fourgram <- TermDocumentMatrix(corpus, control=list(tokenizer = qudratoken))
fm <- rowSums(as.matrix(tdm_for_fourgram))
fourgram <- data.frame(words=names(fm), freq=fm)
fourgram <- fourgram[order(fourgram$freq, decreasing=TRUE),]
top_fourgram <- fourgram[1:15,]
ggplot(top_fourgram, aes(x=reorder(words,freq),y=freq))+geom_bar(stat="identity", fill="purple")+ geom_
```




```
wordcloud(fourgram$words,fourgram$freq, max.words=50,random.order=FALSE,rot.per=0.60,colors=brewer.pal(8
```



Finding word coverage

we will create a helper function to determine the number of words required for the desired percenatge coverage of the words in our corporus

```
woperc <- function(percentage) {
totalwords <- sum(onegram$freq)
percent = 0
cumsum = 0
i = 1
while (percent < percentage)
{
cumsum = cumsum + onegram$freq[i]
percent = cumsum/totalwords
i = i + 1
}
return(i)
}
```

for 50% coverage

```
woperc(0.5)
```

```
## [1] 327
```

for 95 % coverage

```
woperc(0.95)
```

```
## [1] 9402
```

Modelling

Next we will create a very simple model, in which use twogram, threegram and fourgrams to predict the next word. for this we will create two new columns in all of our n grams - n-1 gram (n gram minus the last word) - last word (last word in the n gram)

for this purpose, first we will create a helper function to add these column in our N-gram

```
createNgramTable <- function(x){
z <- strsplit(as.character(x$words)," ")
x$nminusgram <- NA
x$lastword <- NA
for (i in 1:nrow(x)){
wo <- vector()
for (j in 1:(length(z[[i]])-1)){
wo <- c(wo,z[[i]][j])
}
x$nminusgram[i] <- paste(wo, collapse=" ")
x$lastword[i] <- tail(z[[i]],1)
}
return(as.data.frame(x, row.names =NULL, stringsAsFactors=FALSE))
}
```

Now we can get our final N-gram model

```
twogramTable <- createNgramTable(twogram)
threegramTable <- createNgramTable(threegram)
fourgramTable <- createNgramTable(fourgram)
```

Prediction Model

we will use stupid backoff method, i.e. we will first check for the n-1 gram column in fourgram then threegram and then twogram, if nothing is matched we will return "the" as our predicted word other wise we will return the lastword column

```
prediction_model <- function(x,y,z,k){
  t<- tolower(x)
  m<- paste(tail(unlist(strsplit(t,' ')),3), collapse=" ")
  u<- paste(tail(unlist(strsplit(t,' ')),2), collapse=" ")
  v<- paste(tail(unlist(strsplit(t,' ')),1), collapse=" ")
  if (stri_count_words(x)>2){
    if (m %in% y$nminusgram){
    i <- y %>% filter(nminusgram==m) %>% .$lastword
      return(i[1])
    } else if (u %in% z$nminusgram){
    i <- z %>% filter(nminusgram==u) %>% .$lastword
        return(i[1])
    } else if (v %in% k$nminusgram){
    i <- k %>% filter(nminusgram==u) %>% .$lastword
        return(i[1])
```

```
      } else {return('the')}
  } else if(stri_count_words(x)==2){
    if (u %in% z$nminusgram){
    i <- z %>% filter(nminusgram==u) %>% .$lastword
        return(i[1])
    } else if (v %in% k$nminusgram){
      i <- k %>% filter(nminusgram==u) %>% .$lastword
        return(i[1])
      } else {return('the')}
  } else if(stri_count_words(x)==1){
    if (v %in% k$nminusgram){
      i <- k %>% filter(nminusgram==u) %>% .$lastword
        return(i[1])
      }else {return('the')}
  } else {print('wrong input')}
}
```

Testing

```
prediction_model("there are", fourgramTable,threegramTable,twogramTable)
```

```
## [1] "some"
```

```
prediction_model("in the middle ", fourgramTable,threegramTable,twogramTable)
```

```
## [1] "of"
```

Nice :)