Data Science Capstone Milestone Report

Rupender Raj Surender Raj

11/3/2020

Synopsis

Here i am performing exploratory data analysis for few text file using NLP and data mining with libraries like "tm", and "RWeka". From this analysis we can understanding the distribution of words and relationship between the word sand later can used latter to develop next word predicting app.

Setting a Seed

To make sure this notebook is reproducible:

```
#set seed
set.seed(1000)
```

Library's

```
library(tm)
library(RWeka)
library(ggplot2)
library(knitr)
library(dplyr)
library(plyr)
library(data.table)
```

Function's

```
# words without contraction form
word_contraction <- function(w){
                rows <- nrow(contr_wo_ls)
                for(i in 1 : rows ){
                 w <- gsub(contr_wo_ls[i,1], contr_wo_ls[i,2], w, perl=T)
                }
        return(w)
}

# function to determine file size
Size_MB <- function(x){round(file.info(x)$size/1024^2)}

# function to determine word count
Word_Count <- function(x){sum(sapply(strsplit(x, " "), length))}

# function to determine line count
Line_Count <- function(x){length(x)}
```

```
# function to Plot
PT <- function(x, y, z, z1, z2 = 0){
        gg <- ggplot(x, aes(File, x[,y], fill = File))+ geom_bar(stat =
                "identity",width = 0.4)+ theme(legend.position="none")+
                ggtitle(z1)+ xlab("Files")+
                ylab(z)+ theme(plot.title = element_text(color="steelblue"))
                if (z2 == 1){
                        gg <- gg + facet_wrap(as.factor(x[,5]))
                }
        return(gg)
}

PT1 <- function(z, z1){
        gg <- ggplot(head(z,20), aes(x = reorder(word,-freq), y = freq))+ geom_bar(stat =
                "identity",width = 0.4,fill="steelblue")+ theme(legend.position="none")+
                ggtitle(z1)+ xlab("Words")+ ylab("Frequency")+
                theme(axis.text.x = element_text(angle = 90, hjust = 1),
                      plot.title = element_text(color="steelblue"))
        return(gg)
}
# function to crate a sample
Text_Sample <- function(x,y){sample(x, length(x) * y)}

# function to write to txt
Text_Out <- function(x,y){writeLines(x, y)}

# function to corpora file
Text_corp <- function(x){VCorpus(VectorSource(x))}

# Replacement function
Repl_func <- content_transformer(function(x,pattern){return(gsub(pattern, "",x))})

# function for Tokenization
Text_t <- function(x){
        Text_c <- tm_map(x, Repl_func, '"')
        Text_c <- tm_map(Text_c, Repl_func, '"')
        Text_c <- tm_map(Text_c, Repl_func, '-')
        Text_c <- tm_map(Text_c, Repl_func, '- ')
        Text_c <- tm_map(Text_c, Repl_func, "@[^\\s]+")
        Text_c <- tm_map(Text_c, removePunctuation)
        Text_c <- tm_map(Text_c, removeNumbers)
        Text_c <- tm_map(Text_c, removeWords, "profanity_word")
        Text_c <- tm_map(Text_c, content_transformer(tolower))
        Text_c <- tm_map(Text_c, stripWhitespace)
}

# Function to write Tokenized Text
Text_wt <- function (x,y) {
        text_t <- data.frame(text=unlist(sapply(x, `[`, "content")),
                             stringsAsFactors=F)
write.csv(text_t,y, row.names=FALSE)
}
```

```r
# Function to determine the frequency of Words

Freq_fun <- function(tdm){ freq <- sort(rowSums(as.matrix(tdm),na.rm = TRUE),
                                        decreasing = TRUE)
        return(data.frame(word = names(freq), freq = freq))
}

## Using the functions described bellow, we generated bigrams, trigrams .. etc from each sample source

unigram <- function(x) NGramTokenizer(x, Weka_control(min = 1, max = 1))
bigram <- function(x) NGramTokenizer(x, Weka_control(min = 2, max = 2))
trigram <- function(x) NGramTokenizer(x, Weka_control(min = 3, max = 3))
quadgram <- function(x) NGramTokenizer(x, Weka_control(min = 4, max = 4))
pentagram <- function(x) NGramTokenizer(x, Weka_control(min = 5, max = 5))
hexagram <- function(x) NGramTokenizer(x, Weka_control(min = 6, max = 6))


Wordcoverage <- function(x,wordcover){
        nwords <- 0
        coverage <- wordcover*sum(x$freq)
        for (i in 1:nrow(x)) {
                if (nwords >= coverage) {
                        return (i)
                        }
                nwords<-nwords+x$freq[i]
        }
}
```

Task 1 - Getting and cleaning the data

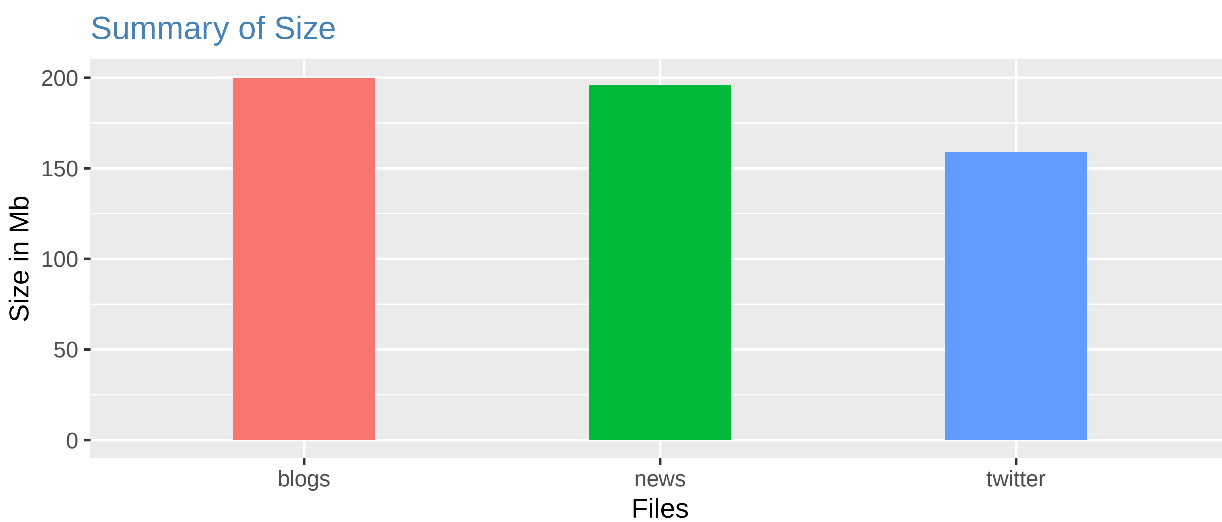
Loading and Reading Data

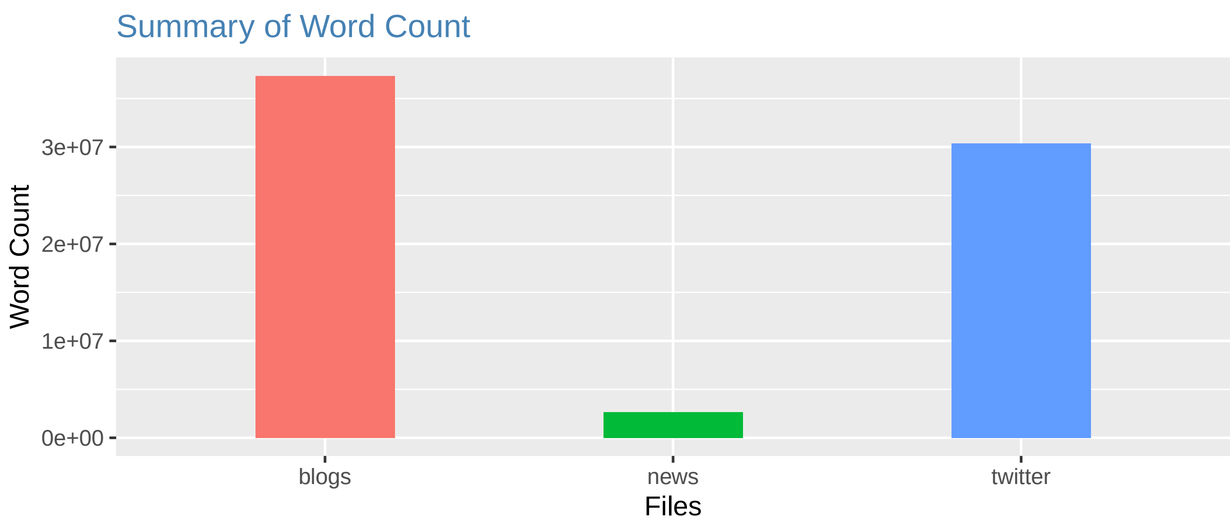
```r
# Working Directory
work_dir <- "F:/DS/ASS/Text_mining"

# Input files
in_dir ="F:/DS/ASS/Text_mining/Input/en_US"

# Read each file in Input Folder
file_list <- "File list"
file_name <- dir(in_dir)
        for (i in 1:3){
        file_seq <- paste(in_dir,file_name[i],sep = "/")
        con <- file(file_seq, "r")
        file_temp <- readLines(con,encoding="UTF-8")
        file_name_temp1 <- unlist(strsplit(file_name[i], ".txt"))
        file_name_temp1 <- strsplit(file_name_temp1, "\\.")[[1]]
        file_name_temp1 <- unique(tolower(file_name_temp1))[2]
        do.call("<-",list(file_name_temp1,file_temp))
        file_list <- c(file_list,file_name_temp1)
        close(con)
        }
```

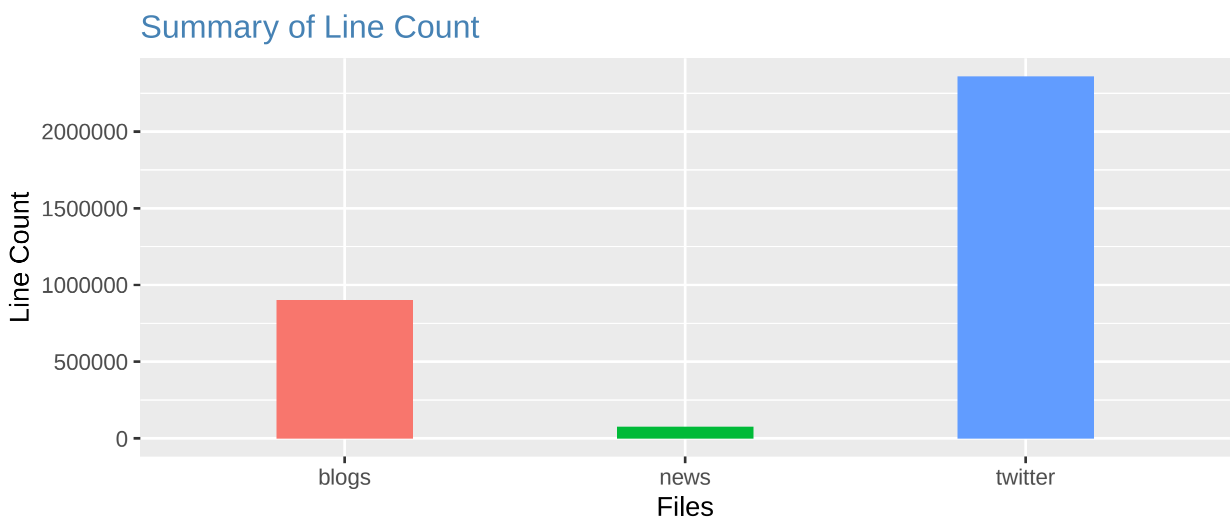
```
Text_summary <- as.data.frame(file_list[2:4])
Text_summary <- cbind(Text_summary,as.data.frame(Size_MB(c(paste(in_dir,file_name[1],
                sep = "/"),paste(in_dir,file_name[2],sep = "/"),
                paste(in_dir,file_name[3],sep = "/")))))
Text_summary <- cbind(Text_summary,as.data.frame(c(Word_Count(blogs),
                Word_Count(news),Word_Count(twitter))))
Text_summary <- cbind(Text_summary,as.data.frame(c(Line_Count(blogs),
                Line_Count(news),Line_Count(twitter))))
File_type <- rep("Orginal file",nrow(Text_summary))
Text_summary <- cbind(Text_summary,File_type)
names(Text_summary) <- c("File","Size MB","Word Count","Line count", "File type")
```

Table 1: Text File Summary

File	Size MB	Word Count	Line count	File type
blogs	200	37334131	899288	Orginal file
news	196	2643969	77259	Orginal file
twitter	159	30373543	2360148	Orginal file







1.Sampling

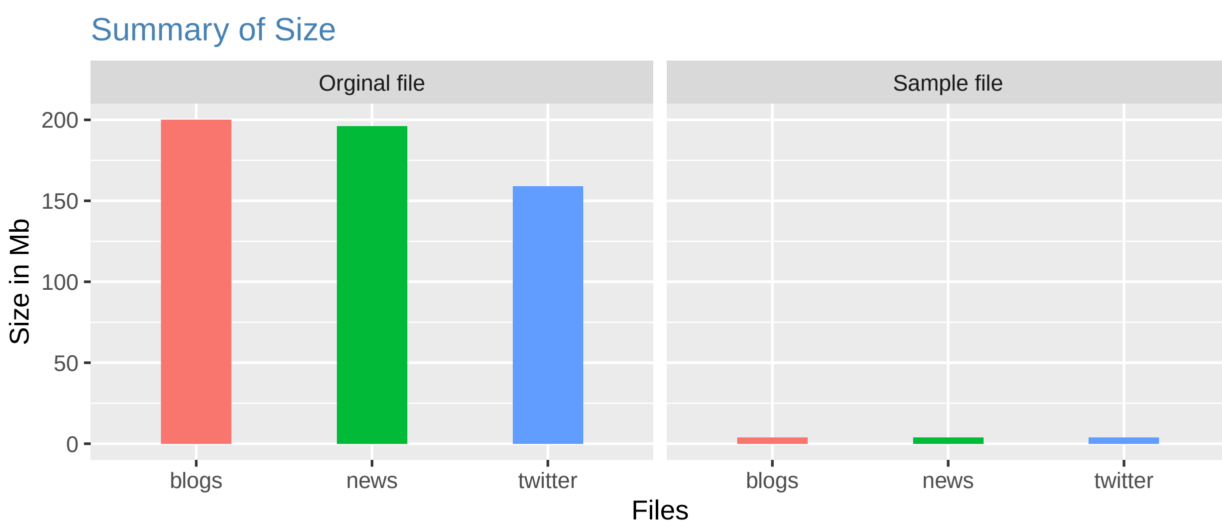
We have three different data files from sources folder. Due to limitations in processing power, a sample of the data is taken. A approx 1800000 consecutive words are considered from the given each data set.

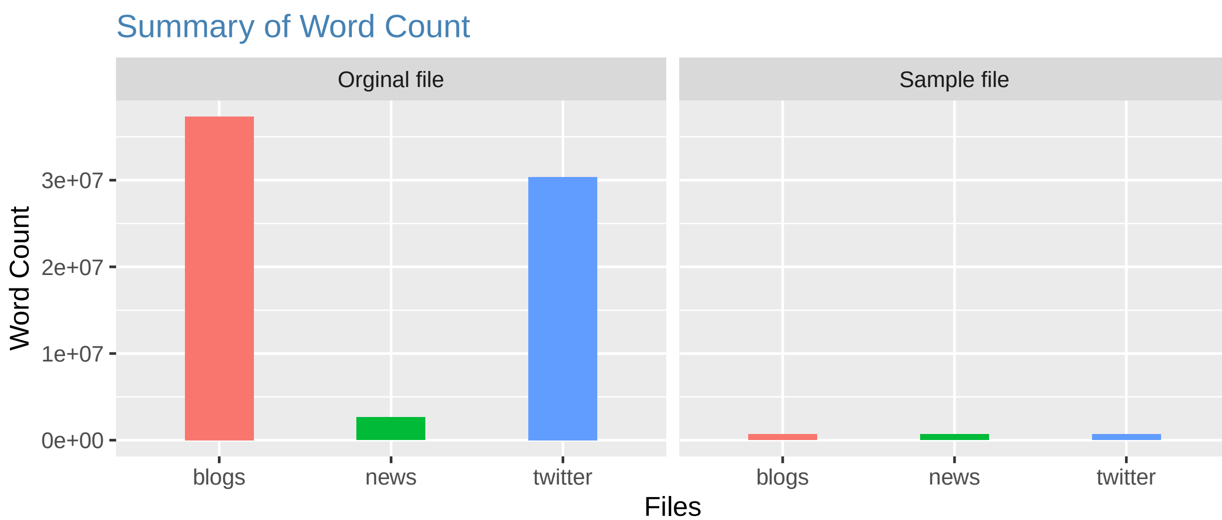
```
blogs_s <- Text_Sample(blogs,0.018) #0.025
news_s <- Text_Sample(news,0.26) #0.35
twitter_s <- Text_Sample(twitter,0.022) #0.03

# words without contraction

contr_wo_ls <- as.data.frame(read.csv("Contraction_en.csv",
                        sep = ',',header = TRUE,))
blogs_s <- word_contraction(blogs_s)
news_s <- word_contraction(news_s)
twitter_s <- word_contraction(twitter_s)
rm(contr_wo_ls)

Text_Out(blogs_s,"blogs_s.txt")
Text_Out(news_s,"news_s.txt")
```

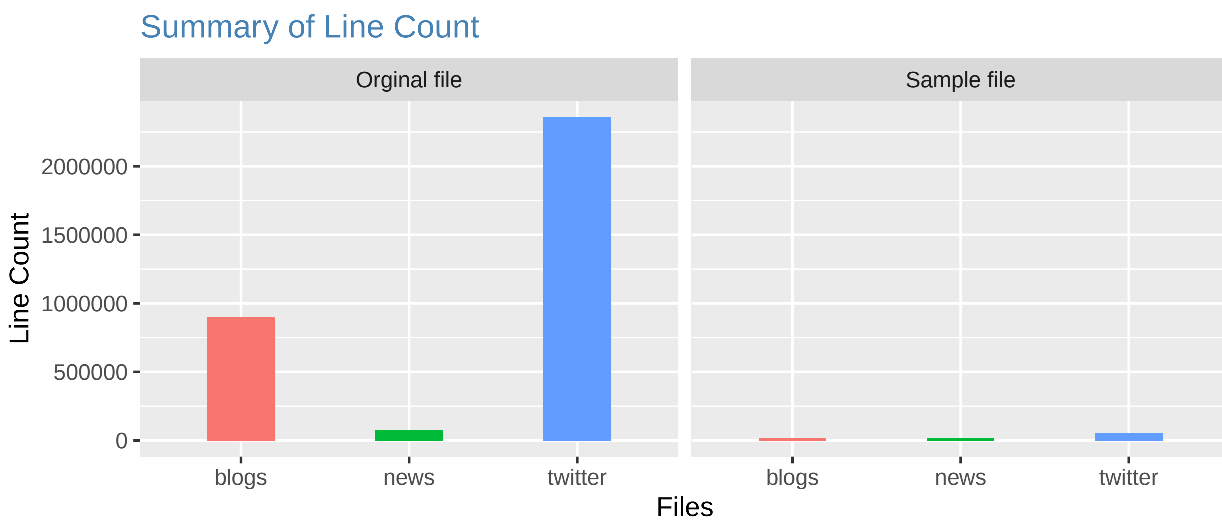
```r
Text_Out(twitter_s,"twitter_s.txt")

Text_summary1 <- as.data.frame(file_list[2:4])
Text_summary1 <- cbind(Text_summary1,as.data.frame(Size_MB(c(paste(work_dir,"blogs_s.txt",
                sep = "/"),paste(work_dir,"news_s.txt",sep = "/"),
                paste(work_dir,"twitter_s.txt",sep = "/")))))
Text_summary1 <- cbind(Text_summary1,as.data.frame(c(Word_Count(blogs_s),
                Word_Count(news_s),Word_Count(twitter_s))))
Text_summary1 <- cbind(Text_summary1,as.data.frame(c(Line_Count(blogs_s),
                Line_Count(news_s),Line_Count(twitter_s))))
File_type <- rep("Sample file",nrow(Text_summary1))
Text_summary1 <- cbind(Text_summary1,File_type)
names(Text_summary1) <- c("File","Size MB","Word Count","Line count", "File type")

Text_summary <- rbind(Text_summary,Text_summary1)
rm(Text_summary1)
```

Table 2: Text File Summary

File	Size MB	Word Count	Line count	File type
blogs	200	37334131	899288	Orginal file
news	196	2643969	77259	Orginal file
twitter	159	30373543	2360148	Orginal file
blogs	4	679897	16187	Sample file
news	4	688779	20087	Sample file
twitter	4	679320	51923	Sample file







From the figure it can be seen the sample data extracted from the original files have very similar in size and Total Word count. hence with sample data we could further clean and Tokenize the data for unnecessary arguments.

2.Tokenization

Now we can token all words associated with text by creating a corpus and then removing bad words, punctuation and numbers. For profanity filtering, we downloaded a "badwords" list (source: https://github.com/shutterstock/List-of-Dirty-Naughty-Obscene-and-Otherwise-Bad-Words/blob/master/en) and removed the words accordingly.

```
# Corpora file Creation
blogs_s_c <- Text_corp(blogs_s)
news_s_c <- Text_corp(news_s)
twitter_s_c <- Text_corp(twitter_s)

# Remove double forward and backward quotes

profanity_word <- read.csv("profanity_words.csv")

blogs_t <- Text_t(blogs_s_c)
```

	word	freq		word	freq		word	freq
the	the	33923	the	the	39349	the	the	20700
and	and	19752	and	and	17662	you	you	12741
that	that	8320	that	that	7065	and	and	9441
for	for	6452	for	for	7019	for	for	8549
you	you	5660	with	with	5127	that	that	5708
with	with	5205	said	said	4943	not	not	5585
was	was	5113	was	was	4576	are	are	4554
this	this	4752	not	not	3943	have	have	4423
have	have	4268	have	have	3213	your	your	3846
not	not	4252	are	are	3162	with	with	3707
are	are	3733	his	his	3126	this	this	3610
but	but	3703	from	from	3015	just	just	3377
from	from	2683	but	but	3004	will	will	3062
all	all	2645	will	will	2504	but	but	2650
they	they	2511	has	has	2486	was	was	2632
will	will	2397	they	they	2446	like	like	2621
one	one	2320	this	this	2446	what	what	2614
had	had	2213	who	who	2236	get	get	2585
about	about	2156	you	you	2180	all	all	2583
his	his	2022	about	about	1820	out	out	2532

x
Word Frequency in Blogs,News and Twitter text File

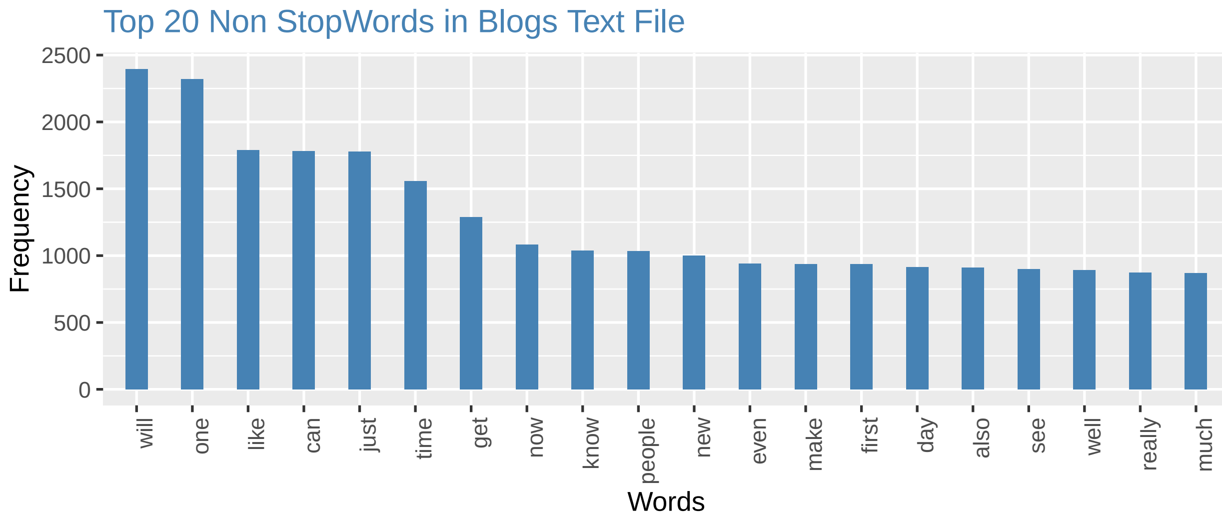
```r
news_t <- Text_t(news_s_c)
twitter_t <-Text_t(twitter_s_c)

Text_wt(blogs_t,"blogs_t.txt")
Text_wt(news_t,"news_t.txt")
Text_wt(twitter_t,"twitter_t.txt")
```

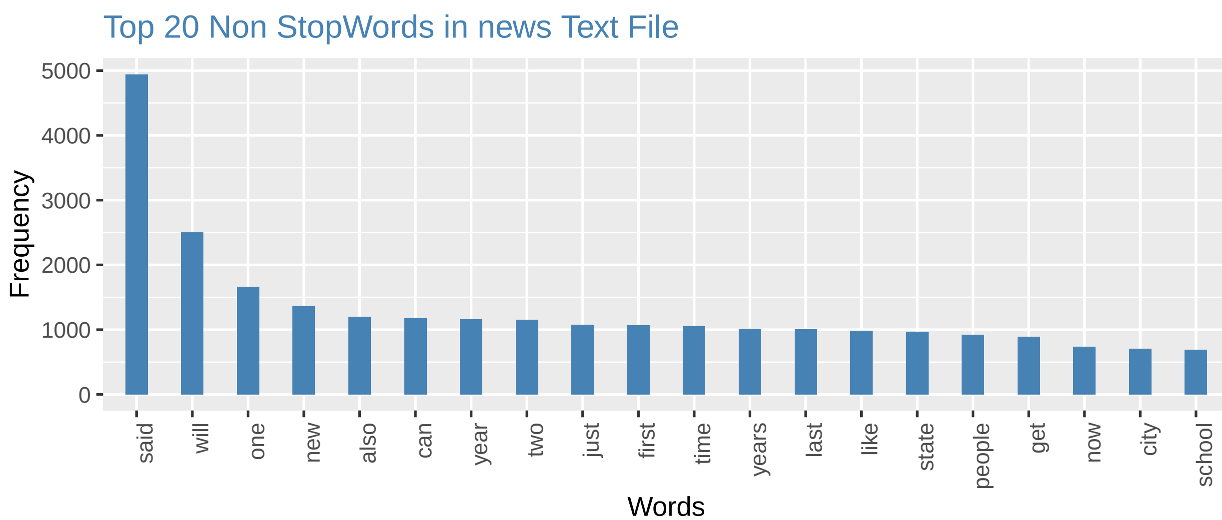
Task 1 - Exploratory Data Analysis

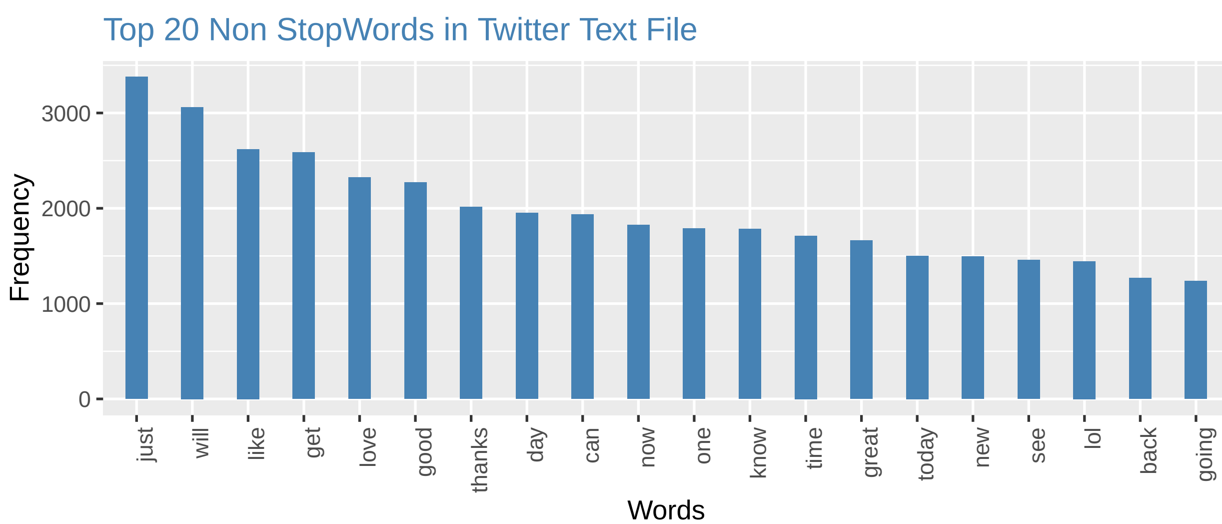
To understanding the distribution of words and relationship between the words in the corpora, we will use TermDocumentMatrix

```r
blogs_tdm <- TermDocumentMatrix(blogs_t)
news_tdm <- TermDocumentMatrix(news_t)
twitter_tdm <- TermDocumentMatrix(twitter_t)

#  Word frequencies

blogs_wf <- Freq_fun(removeSparseTerms(blogs_tdm, 0.999))
news_wf <- Freq_fun(removeSparseTerms(news_tdm, 0.999))
twitter_wf <- Freq_fun(removeSparseTerms(twitter_tdm, 0.999))
```

From the Table 3 its clear that most frequent words are Stop words of English. let see what are the other words other than Stop words







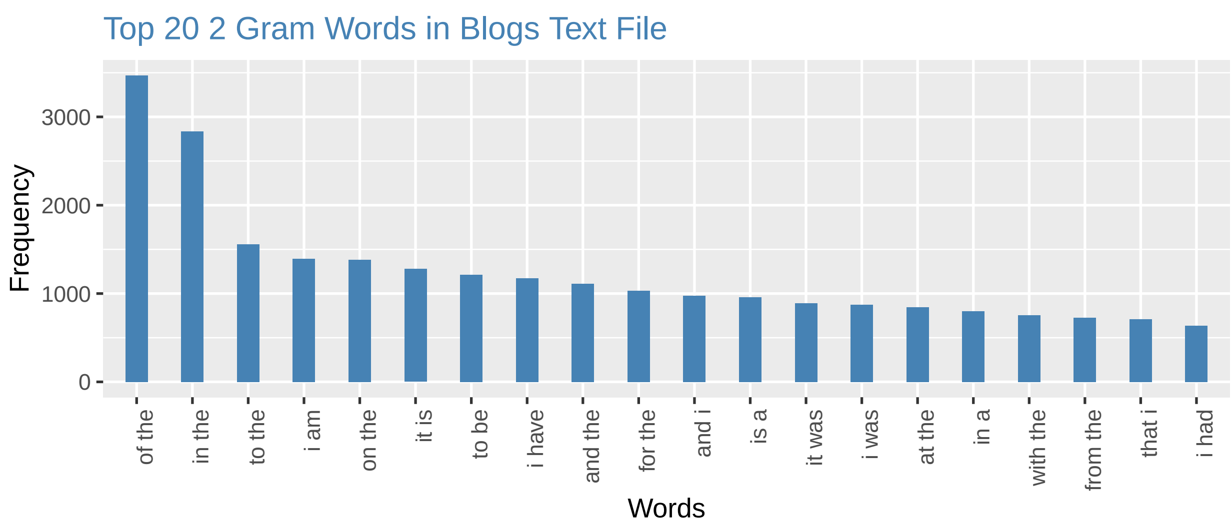
The figure show the frequencies of words other than Stop words.

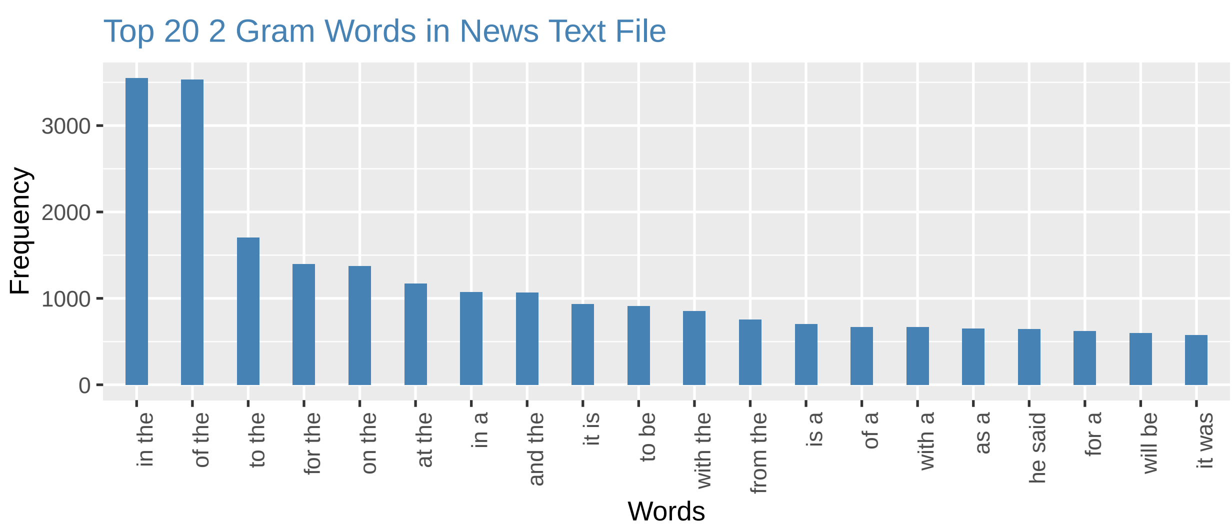
Understand frequencies of words and word pairs

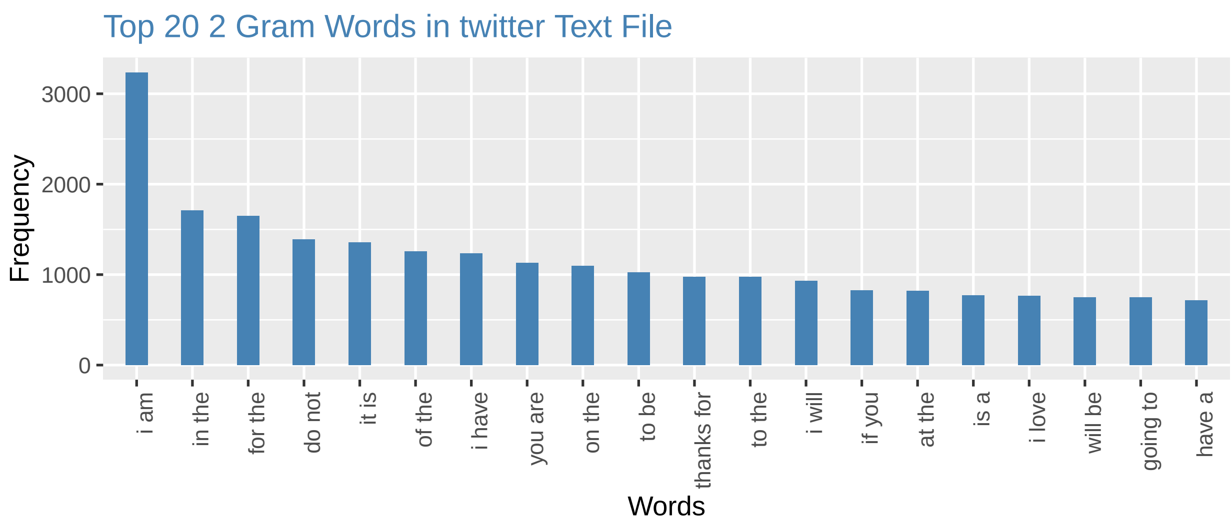
Frequencies of 2-Grams

```
# 2 - grams
blogs_2gram <- Freq_fun(TermDocumentMatrix(blogs_t,
        control = list(tokenize = bigram, bounds = list(global = c(50, Inf)))))
news_2gram <- Freq_fun(TermDocumentMatrix(news_t,
        control = list(tokenize = bigram, bounds = list(global = c(50, Inf)))))
twitter_2gram <- Freq_fun(TermDocumentMatrix(twitter_t,
        control = list(tokenize = bigram, bounds = list(global = c(50, Inf)))))
```

2- Gram Words Frequency



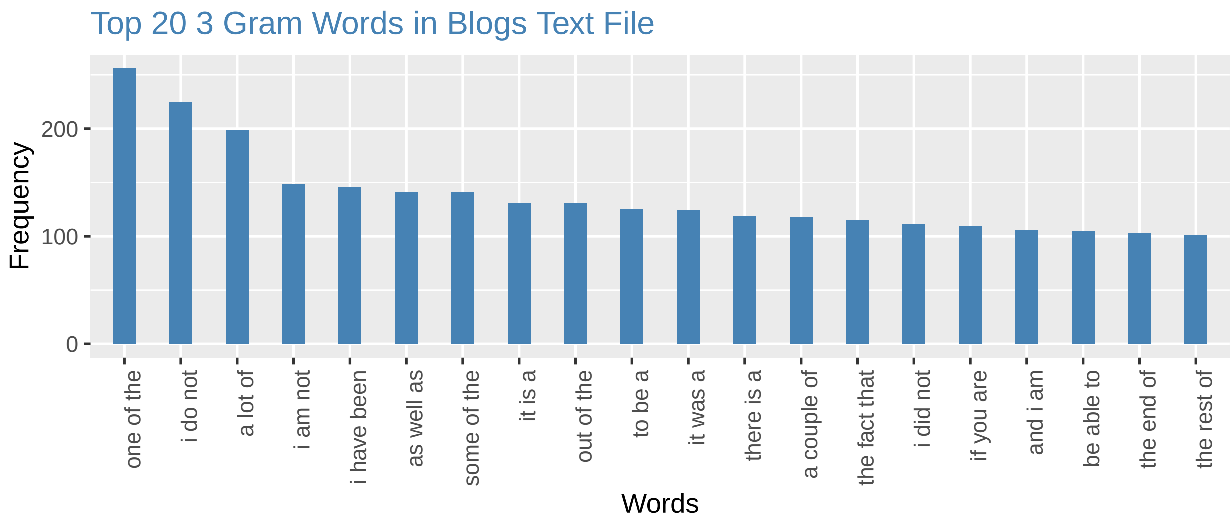


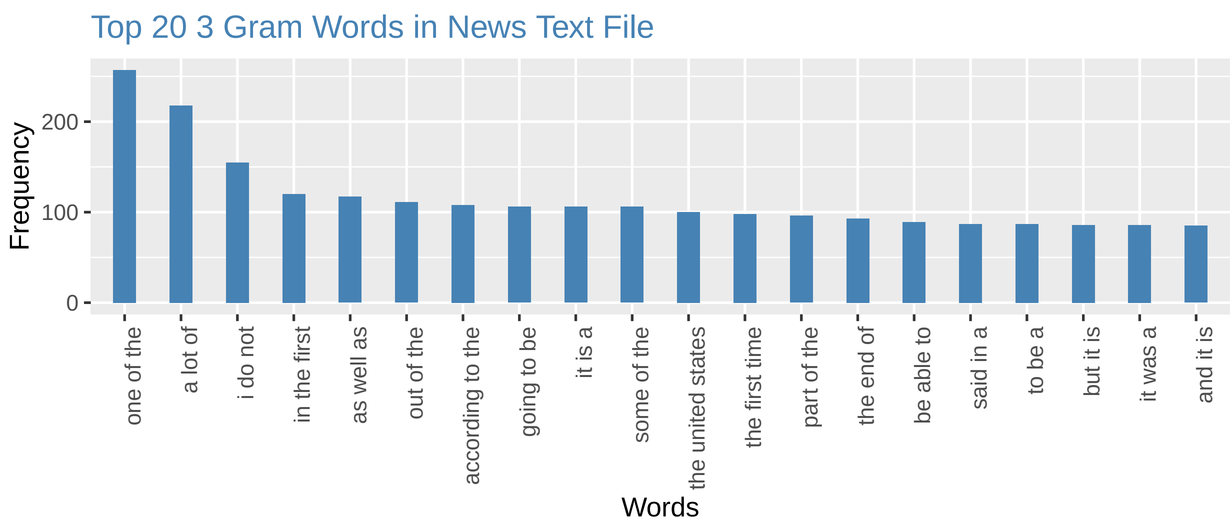


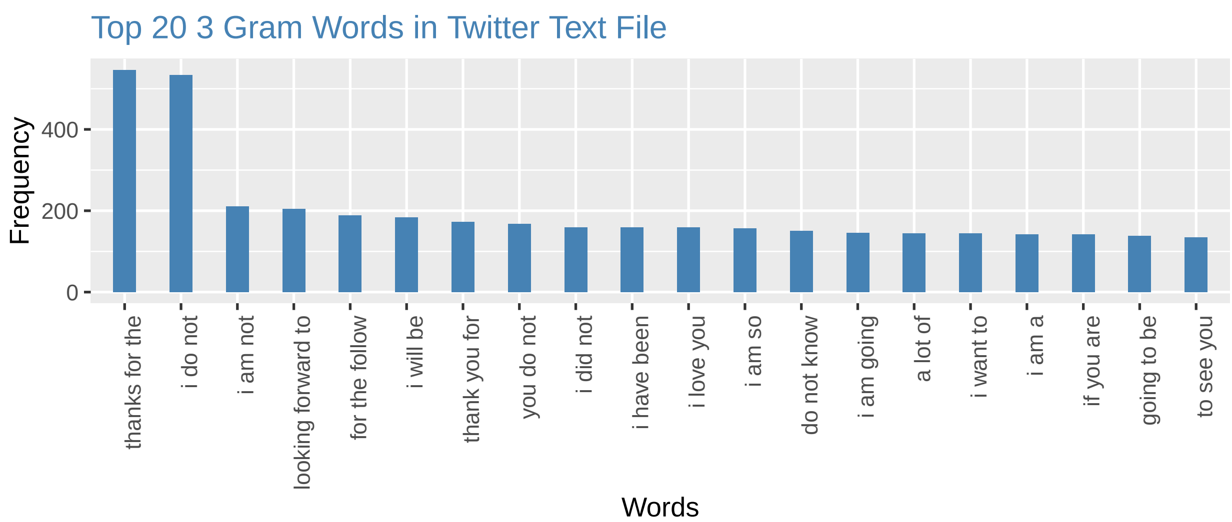
Frequencies of 3-Grams

```
blogs_3gram <- Freq_fun(TermDocumentMatrix(blogs_t,
        control = list(tokenize = trigram, bounds = list(global = c(30, Inf)))))
  news_3gram <- Freq_fun(TermDocumentMatrix(news_t,
        control = list(tokenize = trigram, bounds = list(global = c(30, Inf)))))
twitter_3gram <- Freq_fun(TermDocumentMatrix(twitter_t,
        control = list(tokenize = trigram, bounds = list(global = c(30, Inf)))))
```

3- Gram Words Frequency







Unique Word Coverage

Unique words needed in a frequency sorted dictionary to cover 50% of all word instances in the language? and 90%?

```
# Merge the Text Files

All_Text <- c(blogs_t,news_t,twitter_t)
All_Text_tdm <- TermDocumentMatrix(All_Text)
All_Text_1gram <- Freq_fun(removeSparseTerms(All_Text_tdm, 0.999))

# 50% Word Coverage
Wordcoverage(All_Text_1gram,0.5)
```

```
## [1] 94
```

```
# 50% Word Coverage
Wordcoverage(All_Text_1gram,0.9)
```

```
## [1] 1144
```

Unsurprisingly, the number of words increases exponentially when we increase our desired percent coverage of the language. This is because the frequency of unique words appearing in the corpora also drop exponentially. Hence, for a higher word coverage of the language or dictionary, it will require an exponential increase in number words.

Foreign Language Evaluation

The code developed in this exploratory analysis is not discriminating of languages. When it is necessary to evaluate words from foreign languages, one can make use of the "tm_map" function to "removeWords" based on a language dictionary. The difference in word count will provide insight into the number of words from that particular language in the corpora.

Increasing Coverage

There are several ways that could be used to increase the coverage. One is to reduce the number of low-frequency unique words by stemming or by substitution using a thesaurus library. Additionally, increasing the coverage is possible via context-clustering - with the introduction of a context to the corpora, it will be possible to cluster certain word groups together. For example, if the snapshot of the twitter corpora is taken during a major sporting event, there are many terms, lingos and slangs that could be clustered within the context.

Task 3 - Modeling

- To Build model predicting the next word based on the previous 1, 2, or 3 words we can use N-gram models.
- To handle unseen n-grams in N-gram we can use Katz Backoff Mode.
- And finaly Markov chain is used to store the model effeciently.

Building N-gram Frequencies

```
corpus_without_curse_words <- All_Text
# N-grams of different sizes Function
```

```r
grams1 <- Freq_fun(TermDocumentMatrix(All_Text,
        control = list(tokenize = unigram, bounds = list(global = c(50, Inf)))))
grams2 <- Freq_fun(TermDocumentMatrix(All_Text,
        control = list(tokenize = bigram, bounds = list(global = c(40, Inf)))))
grams3 <- Freq_fun(TermDocumentMatrix(All_Text,
        control = list(tokenize = trigram, bounds = list(global = c(15, Inf)))))
grams4 <- Freq_fun(TermDocumentMatrix(All_Text,
        control = list(tokenize = quadgram, bounds = list(global = c(5, Inf)))))
grams5 <- Freq_fun(TermDocumentMatrix(All_Text,
        control = list(tokenize = pentagram, bounds = list(global = c(3, Inf)))))

ngrams <- rbind(grams1,grams2,grams3,grams4,grams5) %>%
        arrange((word))
frequencies_dt <- ngrams
colnames(frequencies_dt) <- c("ngram","frequency")
```

Table 3: N-gram Frequencies

ngram	frequency
i do not have any	5
theres a lot of	9
of what i	20
concert	150
was the same	15
of passes for yards	5
look forward to a	7
morning i	46
said i do not want	3
produced	87
will find	55
have so many	21

The Table Show the 12 random N-grams and its frequencies.

An history column is created in the table

```r
# function to calculate words history
extract_history <- function(ngram){
    ifelse(length(ngram) > 1,
           paste(ngram[1:(length(ngram)-1)], collapse = " "),
           ""
    )
}

extract_word <- function(ngram) paste(ngram[length(ngram)], collapse = " ")

build_processed_ngram_frequencies <- function(frequencies_dt) {
    data <- as.data.frame(frequencies_dt)
    data$ngram <- strsplit(data$ngram, split = " |'")
    data$ngram_length <- sapply(data$ngram, length)
    data$history <- sapply(data$ngram, extract_history)
    data$word <- sapply(data$ngram, extract_word)
    data$ngram <- sapply(data$ngram, paste, collapse = " ")
```

```r
    data.table(data)
}
# Extract the history word
ngram_frequencies_dt <- build_processed_ngram_frequencies(frequencies_dt)
```

Table 4: N-gram Frequencies with its word history

ngram	frequency	ngram_length	history	word
of the	8263	2	of	the
in the	8100	2	in	the
i am	5068	2	i	am
to the	4238	2	to	the
for the	4076	2	for	the
on the	3851	2	on	the
it is	3564	2	it	is
to be	3145	2	to	be
at the	2836	2	at	the
i have	2683	2	i	have
and the	2519	2	and	the
do not	2482	2	do	not
is a	2434	2	is	a
in a	2351	2	in	a
with the	1944	2	with	the
it was	1942	2	it	was
will be	1915	2	will	be
for a	1902	2	for	a
and i	1836	2	and	i
you are	1771	2	you	are
from the	1737	2	from	the
is the	1713	2	is	the
if you	1673	2	if	you
i was	1657	2	i	was
going to	1583	2	going	to
with a	1575	2	with	a
of a	1519	2	of	a
i will	1517	2	i	will
that is	1464	2	that	is
to get	1428	2	to	get
have a	1422	2	have	a
as a	1394	2	as	a
one of	1376	2	one	of
is not	1337	2	is	not
did not	1284	2	did	not
i do	1254	2	i	do
have been	1244	2	have	been
want to	1244	2	want	to
i had	1241	2	i	had
have to	1215	2	have	to
by the	1206	2	by	the
this is	1200	2	this	is
but i	1182	2	but	i
to do	1178	2	to	do
the first	1152	2	the	first

ngram	frequency	ngram_length	history	word
we are	1152	2	we	are
that the	1149	2	that	the
i think	1138	2	i	think
that i	1136	2	that	i
and a	1106	2	and	a

To computing probabilities for the model will require counts for histories as well, so let's create a data table specifically for this purpose

```
history_frequencies_dt <-
    ngram_frequencies_dt[, c("history", "frequency")][, lapply(.SD, sum), by = list(history)]
```

Table 5: Frequency of History Words

history	frequency
	1245903
the	38544
to	34405
i	31524
in	19484
and	19400
of	19128
a	17409
is	15850
it	13050
you	12860
for	12567
that	10852
on	9891
with	7364
not	7232
have	7172
at	6341
as	5842
but	5582
we	5557
do	5268
was	4989
he	4934
are	4817
this	4681
will	4442
if	4328
be	3979
so	3969
what	3902
they	3742
my	3623
out	3546
when	3493
me	3455
all	3371

history	frequency
from	3289
in the	3143
up	3103
i am	3029
there	2838
like	2827
about	2820
am	2723
said	2635
one	2621
get	2596
had	2416
has	2374

```
frequencies_of_frequencies <- table(ngram_frequencies_dt$frequency)
frequencies_of_frequencies
```

```
##
##    3    4    5    6    7    8    9   10   11   12   13   14   15
## 1412  478 1677  913  605  369  278  186  143  119   89   77  401
##   16   17   18   19   20   21   22   23   24   25   26   27   28
##  369  315  268  226  182  223  180  147  133  117  119  113   84
##   29   30   31   32   33   34   35   36   37   38   39   40   41
##   75   76   89   81   60   60   49   43   34   42   33   98  125
##   42   43   44   45   46   47   48   49   50   51   52   53   54
##  109  117  135  126  124  101   95  103  108   99   88  121  116
##   55   56   57   58   59   60   61   62   63   64   65   66   67
##  108  121  129  102  126   92  113  112   90   84   86   86   74
##   68   69   70   71   72   73   74   75   76   77   78   79   80
##   72   81   66   63   67   60   62   79   66   64   50   58   59
##   81   82   83   84   85   86   87   88   89   90   91   92   93
##   74   50   49   57   50   44   41   49   44   43   45   41   40
##   94   95   96   97   98   99  100  101  102  103  104  105  106
##   44   50   38   35   37   44   41   38   36   35   35   21   30
##  107  108  109  110  111  112  113  114  115  116  117  118  119
##   32   29   40   26   30   27   28   31   24   30   28   31   21
##  120  121  122  123  124  125  126  127  128  129  130  131  132
##   23   21   26   16   29   25   27   23   21   21   25   24   18
##  133  134  135  136  137  138  139  140  141  142  143  144  145
##   23   22   26   20   14   22   20   21   19   15   17   24   23
##  146  147  148  149  150  151  152  153  154  155  156  157  158
##   21   15   19   16   15   10   13   16   17   15    8   15   19
##  159  160  161  162  163  164  165  166  167  168  169  170  171
##    5   10   11   10   14   18   10   15   12   12   16   11   20
##  172  173  174  175  176  177  178  179  180  181  182  183  184
##   19    9    7   13    4   14    8   13    6   13   11    8   17
##  185  186  187  188  189  190  191  192  193  194  195  196  197
##    9   12   12   10   10    8   15   15   18   12   12    9   10
##  198  199  200  201  202  203  204  205  206  207  208  209  210
##    6    3    5    9   10   12   12    6   13    8    4   15   13
##  211  212  213  214  215  216  217  218  219  220  221  222  223
##    9    6   10    8    8    9   12   12    8    8    8    5    7
```

```
##  224  225  226  227  228  229  230  231  232  233  234  235  236
##   12    8    7    6   10    5    4    4    3    8    8    6    8
##  237  238  239  240  241  242  243  244  245  246  247  248  249
##   17    8    7    6    2    8    2   10    2    8   10    3    7
##  250  251  252  253  254  255  256  257  258  259  260  261  262
##    5    9    6    7    3    8    4    6    7    4    9    6    6
##  263  264  265  266  267  268  269  270  271  272  273  274  275
##    7    4    5    3    4    1    5    4    5    4    2    5    3
##  276  277  278  279  280  281  282  283  284  285  286  287  288
##    3    3    5    7    6    4    9    6    5    6    2    1    4
##  289  290  291  292  293  294  295  296  297  298  299  300  301
##    3    4    5    2    8    5    1    5    2    7    3    2    2
##  302  303  304  305  306  307  308  309  310  311  312  313  314
##    2    3    3    2    3    2    4    4    1    4    4    6    6
##  315  316  317  318  319  320  321  322  323  324  325  326  327
##    4    3    2    3    2    3    6    2    2    1    8    5    3
##  328  329  330  331  332  333  334  335  336  337  338  339  340
##    5    5    3    4    1    5    4    3    2    4    2    2    3
##  341  343  344  345  346  347  348  349  350  351  352  353  354
##    3    3    3    2    1    5    2    1    5    4    4    3    4
##  355  356  357  359  360  361  362  363  364  366  367  368  369
##    3    3    3    2    3    2    3    2    4    5    2    1    5
##  370  371  372  373  375  376  377  378  379  380  381  382  383
##    4    3    1    2    2    1    1    2    2    4    1    1    3
##  384  385  386  387  388  389  390  391  392  393  394  395  396
##    2    2    1    1    3    2    2    3    1    6    1    3    2
##  397  398  400  401  403  404  405  406  407  409  411  412  413
##    3    6    1    2    1    3    3    1    3    3    2    4    1
##  414  415  416  417  418  419  420  421  422  423  424  425  426
##    5    2    2    2    1    2    2    1    1    5    3    2    2
##  427  428  429  430  433  434  435  436  438  442  443  444  445
##    3    3    1    6    2    3    4    1    2    1    2    2    2
##  446  448  449  450  451  453  454  455  456  457  458  460  461
##    2    1    2    2    4    2    1    1    2    1    1    1    1
##  462  464  466  467  468  469  470  471  472  473  475  476  477
##    2    2    1    3    1    1    3    1    3    1    5    1    1
##  480  481  482  484  485  486  487  490  491  492  493  494  495
##    2    1    1    1    1    4    3    4    1    1    1    2    1
##  497  499  500  502  503  504  505  506  508  509  511  512  515
##    1    2    2    1    3    1    2    2    2    2    3    2    3
##  516  519  520  521  522  525  526  528  529  530  531  532  534
##    4    2    2    1    1    1    1    4    2    2    1    4    1
##  538  539  541  542  543  547  548  549  550  551  552  553  554
##    2    1    1    3    2    1    3    1    1    2    1    1    2
##  555  556  558  559  560  561  562  565  566  569  572  575  576
##    1    1    1    2    1    2    1    1    1    1    1    1    1
##  577  579  580  581  582  584  585  586  587  589  590  591  592
##    1    2    1    2    1    1    1    1    1    1    1    3    1
##  594  596  598  599  600  602  603  604  606  607  610  614  615
##    1    1    1    1    2    2    1    2    2    2    1    1    2
##  616  617  620  621  622  623  624  625  626  628  630  635  636
##    1    2    1    2    1    1    2    1    1    2    2    1    1
##  637  638  639  641  646  647  648  651  653  654  658  660  661
##    1    1    2    3    1    2    1    1    1    2    1    1    1
```

```
##   662   663   664   665   666   670   671   675   678   679   680   683   684
##     2     1     2     1     2     1     1     1     1     2     1     1     1
##   687   688   690   693   696   698   699   701   702   704   706   707   713
##     1     1     1     1     1     2     1     1     3     2     1     1     2
##   716   717   718   719   723   724   727   733   734   735   737   739   742
##     1     1     1     2     1     1     1     2     1     1     1     1     1
##   743   744   746   747   748   752   753   754   757   762   767   772   773
##     2     1     1     1     1     1     1     1     1     1     1     1     1
##   774   781   782   784   792   796   799   800   804   810   813   815   819
##     1     1     1     1     1     1     3     1     1     1     1     2     1
##   820   823   827   829   832   833   838   839   840   841   843   845   847
##     2     1     1     1     1     2     1     1     2     1     2     2     1
##   851   852   856   857   859   864   865   868   873   876   878   881   882
##     2     1     1     1     2     2     2     1     1     1     1     1     1
##   887   889   891   892   894   895   906   910   912   914   917   919   920
##     1     1     1     1     1     1     1     1     2     3     1     1     1
##   925   956   967   982   985   993  1002  1005  1008  1011  1014  1015  1016
##     1     1     2     1     1     1     1     1     2     1     2     1     1
##  1023  1024  1032  1033  1036  1039  1043  1056  1058  1060  1070  1073  1074
##     1     1     1     1     1     1     1     1     1     1     1     1     1
##  1075  1078  1083  1095  1100  1102  1103  1106  1113  1121  1126  1129  1132
##     1     1     1     1     1     2     1     1     1     1     1     1     1
##  1133  1136  1138  1141  1149  1152  1155  1177  1178  1182  1183  1188  1200
##     1     1     1     1     1     2     1     1     1     2     1     1     1
##  1202  1206  1212  1215  1226  1241  1244  1246  1254  1268  1269  1284  1289
##     1     1     1     1     1     1     2     1     1     1     1     1     1
##  1290  1293  1295  1304  1322  1326  1327  1337  1343  1353  1359  1376  1394
##     1     1     1     1     1     1     1     1     1     1     1     1     1
##  1422  1428  1450  1453  1464  1471  1488  1491  1517  1519  1531  1538  1544
##     1     1     1     1     1     1     1     1     2     1     1     1     1
##  1547  1575  1578  1579  1583  1587  1605  1633  1643  1657  1667  1670  1673
##     1     1     1     1     1     1     1     1     1     1     1     1     1
##  1694  1696  1701  1709  1713  1723  1736  1737  1749  1765  1770  1771  1809
##     1     1     1     1     1     1     1     1     1     1     1     1     1
##  1820  1836  1860  1871  1891  1902  1910  1915  1918  1940  1942  1944  1973
##     1     1     1     1     1     1     1     1     1     1     1     1     1
##  2048  2070  2101  2148  2151  2168  2187  2213  2220  2232  2243  2259  2335
##     1     1     1     1     1     1     1     1     1     1     1     1     1
##  2345  2351  2380  2394  2413  2434  2455  2461  2466  2469  2482  2499  2505
##     1     1     1     1     1     1     1     1     1     1     1     2     1
##  2513  2519  2533  2569  2587  2593  2633  2683  2697  2773  2779  2826  2836
##     1     1     1     1     1     1     1     1     2     1     1     1     1
##  3066  3131  3145  3153  3219  3352  3399  3564  3584  3647  3687  3689  3712
##     1     1     1     1     1     1     1     1     1     1     1     1     1
##  3851  3857  3907  3977  3984  3986  4076  4179  4211  4224  4238  4320  4572
##     1     1     1     1     1     1     1     1     1     1     1     1     1
##  4697  4765  4892  5068  5150  5195  5272  5293  5393  5722  5772  5850  5875
##     1     1     1     1     1     1     1     1     1     1     1     1     1
##  5949  5963  6228  6293  6605  6633  7501  7963  8100  8263  9357 10806 11449
##     1     1     1     1     1     1     1     1     1     1     1     1     1
## 11904 12321 13780 14038 20580 21093 22020 46855 93972
##     1     1     1     1     1     1     1     1     1
```

Good-Turing discount Function

```
frequency_of_frequency <- function(frequency, frequencies_of_frequencies)
    try_default(frequencies_of_frequencies[[toString(frequency)]], 1, quiet = TRUE)

discount <- function(count, frequencies_of_frequencies) {
    good_turing_count <- (count + 1) *
        frequency_of_frequency(count + 1, frequencies_of_frequencies) /
        frequency_of_frequency(count, frequencies_of_frequencies)
    computed_discount <- good_turing_count / count
    ifelse(computed_discount < 1, computed_discount, 1)
}
```

Testing Good- turing for 1 counts is 1

Katz's probability calculation

```
count_ngram <- function(word_value, history_value, ngram_frequencies_dt) {
    count <- ngram_frequencies_dt[word == word_value & history == history_value, ]$frequency
    ifelse(length(count) > 0, count, 1)
}

count_history <- function(history_value, history_frequencies_dt) {
    count <- history_frequencies_dt[history == history_value, ]$frequency
    ifelse(length(count) > 0, count, 1)
}
```

Count of word "the" after word "in" in a sentence is 8100

Frequency of word "in" as precedence word is 1.9484×10^4

```
# Function to remove the first word in a sentences

backoff_history <- function(history) {
    history_words <- strsplit(history, split = " |'")
    ifelse(
        length(history_words[[1]]) > 1,
        trimws(paste(backoff_history_words <- history_words[[1]][2:length(history_words[[1]])], collaps
        ""
    )
}
```

Testing backoff_history Function

In a sentences "bird is the word" the function will remove each word from left 1 step …1st word… is the word 2 step …2 words…. is the word

```
# function to calculate Beta probability
katz_beta <- function(history_value, k,
                      ngram_frequencies_dt,
                      history_frequencies_dt,
                      frequencies_of_frequencies) {

    counts <- ngram_frequencies_dt[history == history_value & frequency > k, ]$frequency
    history_count <- count_history(history_value, history_frequencies_dt)
```

```
    1 - ifelse(length(counts) > 0,
               sum(
                   sapply(counts,
                          function(x) discount(x, frequencies_of_frequencies) * x / history_count
                   )
               ),
               0
    )
}

#memoized_katz_beta <- addMemoization(katz_beta)
```

Testing Katz's beta probability for the word "the" is 0.1401981

```
total_words <- sum(ngram_frequencies_dt[ngram_length == 1, ]$frequency)

katz_alpha_summation <- function(history_value, k,
                                 ngram_frequencies_dt,
                                 history_frequencies_dt,
                                 frequencies_of_frequencies) {
    words <- ngram_frequencies_dt[history == history_value & frequency <= k, ]$word
    ifelse(length(words) > 0,
           sum(
               sapply(words,
                      katz_probability,
                      history = backoff_history(history_value),
                      k = k,
                      ngram_frequencies_dt = ngram_frequencies_dt,
                      history_frequencies_dt = history_frequencies_dt,
                      frequencies_of_frequencies = frequencies_of_frequencies
               )
           ),
           0
    )
}

#memoized_katz_alpha_summation <- addMemoization(katz_alpha_summation)

katz_alpha <- function(history, k,
                       ngram_frequencies_dt,
                       history_frequencies_dt,
                       frequencies_of_frequencies) {

    computed_katz_alpha_summation <- katz_alpha_summation(history,
                          k, ngram_frequencies_dt,
                          history_frequencies_dt,
                          frequencies_of_frequencies)

    computed_katz_beta <- katz_beta(history, k,
                          ngram_frequencies_dt,
                          history_frequencies_dt,
                          frequencies_of_frequencies)

    computed_katz_alpha <- ifelse(computed_katz_alpha_summation !=0,                          computed_
```

```
                        computed_katz_alpha_summation, 1)
    ifelse(computed_katz_alpha < 1, computed_katz_alpha, 1)
}

#memoized_katz_alpha <- addMemoization(katz_alpha)

katz_probability <- function(word, history, k,
                            ngram_frequencies_dt,
                            history_frequencies_dt,
                            frequencies_of_frequencies,
                            verbose = FALSE) {
    if(verbose) print(paste0("katz_probability(word: [", word, "], history: [",
                            history, "])..."))

    word_with_history <- trimws(paste(history, word))
    count <- count_ngram(word, history, ngram_frequencies_dt)

    probability <- ifelse(history == "", discount(count,
                               frequencies_of_frequencies) * count /
                                total_words,
        ifelse(count > k, discount(count, frequencies_of_frequencies) * count /          count_history
         ngram_frequencies_dt, history_frequencies_dt, frequencies_of_frequencies) * katz_probability(w
                                    ngram_frequencies_dt,
                                    history_frequencies_dt,
                                    frequencies_of_frequencies,
                                    verbose = verbose)
        )
    )

    if(verbose) print(paste0("katz_probability(word: [", word, "], history: [", history, "]) = ", proba
    probability
}
```

Testing Katz's alpha simulation probability for the word "the" is 0

Testing Katz's alpha probability for the word "the" is 1

Testing Katz's probability for the word "did" before the word "i" is 1.7205882

Computing a Markov Chain Trainsition Matrix

With Katz's alpha and Beta probability we can know store the Ngrams in more efficient way representing Markov Chain

```
create_transition_matrix <- function(k,
                                     ngram_frequencies_dt,
                                     history_frequencies_dt,
                                     frequencies_of_frequencies,
                                     min_count = 3, verbose = FALSE) {

    prediction_data <- ngram_frequencies_dt[ngram_length > 1 & frequency > min_count, ]

    histories <- unique(c("", prediction_data$history))
    print(paste0("history size: ", length(histories)))
```

```r
    words <- unique(prediction_data$word)
    print(paste0("word size: ", length(words)))

    transition_matrix <- matrix(NA, length(histories), length(words))
    rownames(transition_matrix) <- histories
    colnames(transition_matrix) <- words

    percentage_complete <- 0
    percentage_counter <- 0
    for(i in 1:length(histories)){
        for(j in 1:length(words)){
            transition_matrix[i, j] <- katz_probability(words[[j]],
                                                        histories[[i]],
                                                        k,
                                                        ngram_frequencies_dt,
                                                        history_frequencies_dt,
                                                        frequencies_of_frequencies,
                                                        verbose = verbose)
        }
        percentage_complete <- round((i * length(words) + j + 1) / (length(histories) * length(words)) *
        if(percentage_complete > percentage_counter & percentage_complete < 100) {
            percentage_counter <- percentage_complete
            print(paste0(percentage_counter, "% complete."))
        }
    }
    print("complete.")
    transition_matrix
}
```