Data Science Capstone - Peer Graded Assignment: Milestone Report

Data Science Capstone - Milestone Report

This is document is created to complete the course on Coursera: Data Science Capstone by John Hopkins University. It is part of the Peer-graded Assignments. This report addresses the first assignment of the course, which is given in week 2.

The overall goal of the capstone is to create a predictive text model to suggest relevant completion of text. When someone types:

I went to the

the model should present three options for what the next word might be. For example, the three words might be gym, store, restaurant. In this capstone you will work on understanding and building predictive text models.

The Milestone Report is a intermediate step towards the predictive text model.

Overview

The basic goal of this assignment is to display that I’ve gotten used to working with the data and that I am on track to create a prediction algorithm.

The motivation for this project is to:

1. Demonstrate that I’ve downloaded the data and have successfully loaded it in.
2. Create a basic report of summary statistics about the data sets.
3. Report any interesting findings that you amassed so far.
4. Get feedback on your plans for creating a prediction algorithm and Shiny app.

The report is structured in the following way:

- Data collection
    - Downloading data
    - Loading data in
    - Basic data summary
- Text handling
    - Tokenization
    - Ngrams
- Findings
- Next steps
- Appendix with the code

Data collection

Data source

The data used in this report originates from this link. It contains several .txt files in four different languages including English. The files contain lines text scraped from text sources like blogs, news and twitter.

Downloading data

Loading data in

Basic data summary

The basic information of the different text sources is listed in the table below.

	Size [MB]	No. of lines
blogs	200.4242	899288
news	196.2775	77259
twitter	159.3641	2360148

Text handling

A next step after loading in the data, is to analyze is. As the total data set is rather large, a random selection of the data is sampled. In a later stage of the project, this might be eloborated further on. For now, a selection suffices to get an idea how the data looks like.

The steps including text handling are as follows:

Text data is bundeled in a *corpus*, this is an organized set of texts.

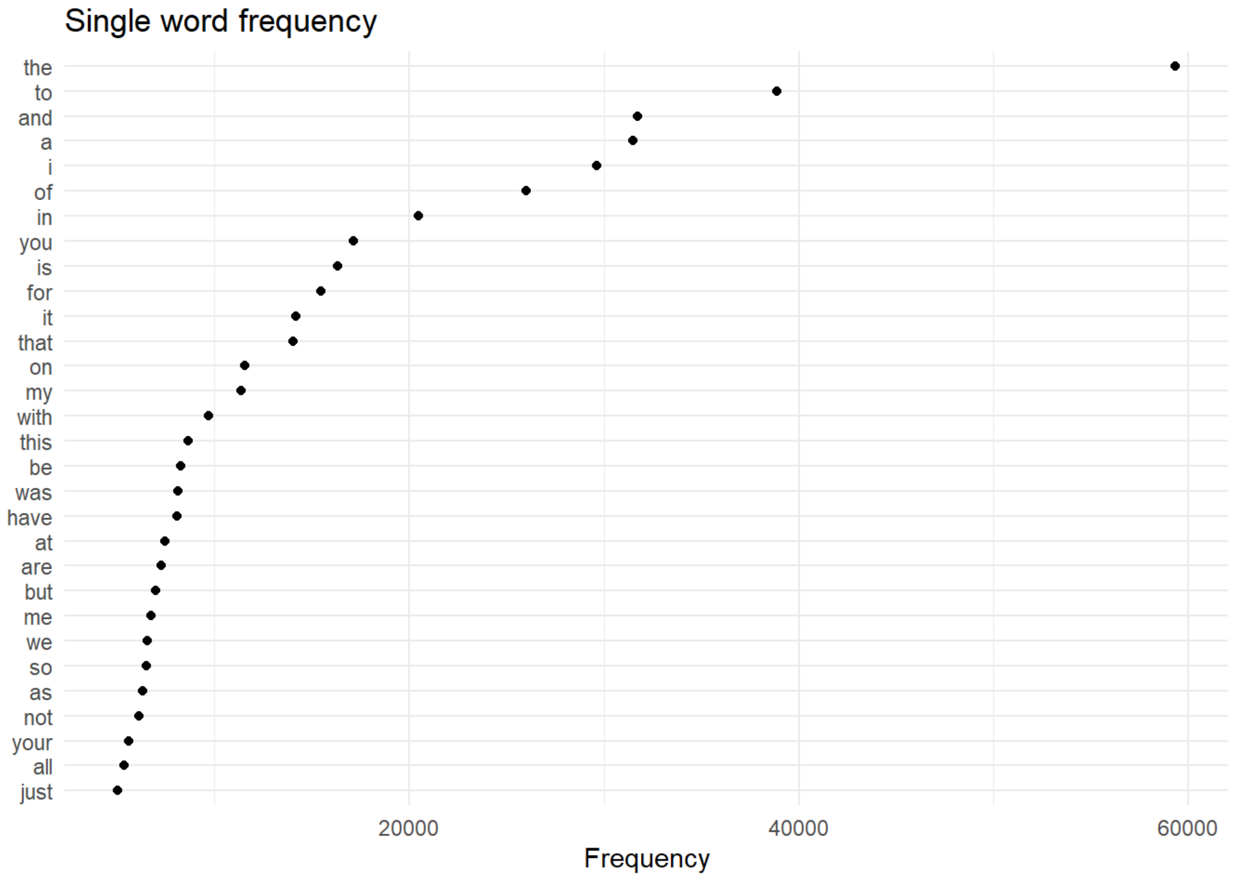
Next, the data is *tokenized*. This means that the raw text lines are stripped into identifying appropriate words or tokens, by removing unwanted characters like symbols. Also a profinaty filter is applied.

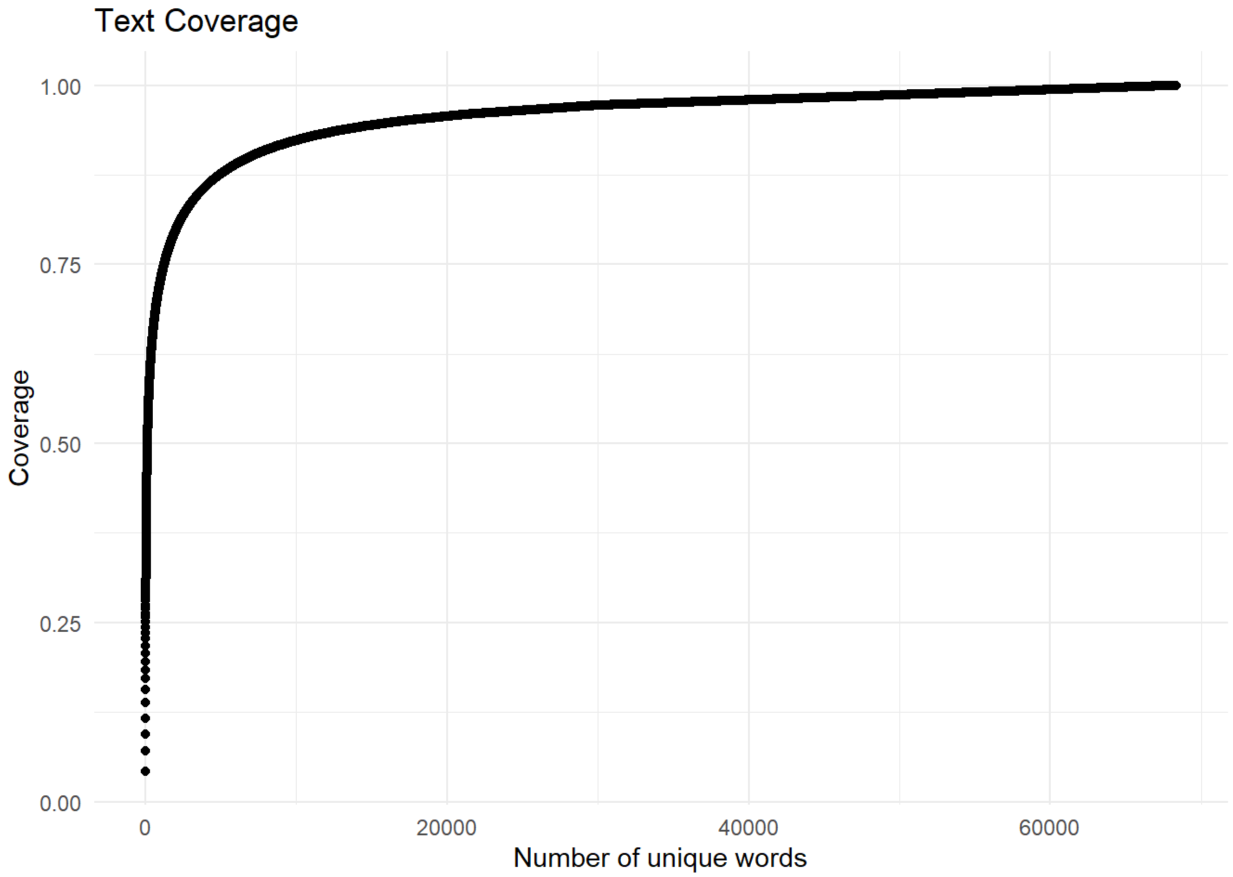
After tokenization, the tokens are analyzed. Special attention is given to the frequencey of the words which are used. For the text prediction model it is also interesting to look at recurring combinations of words: *n-grams*.

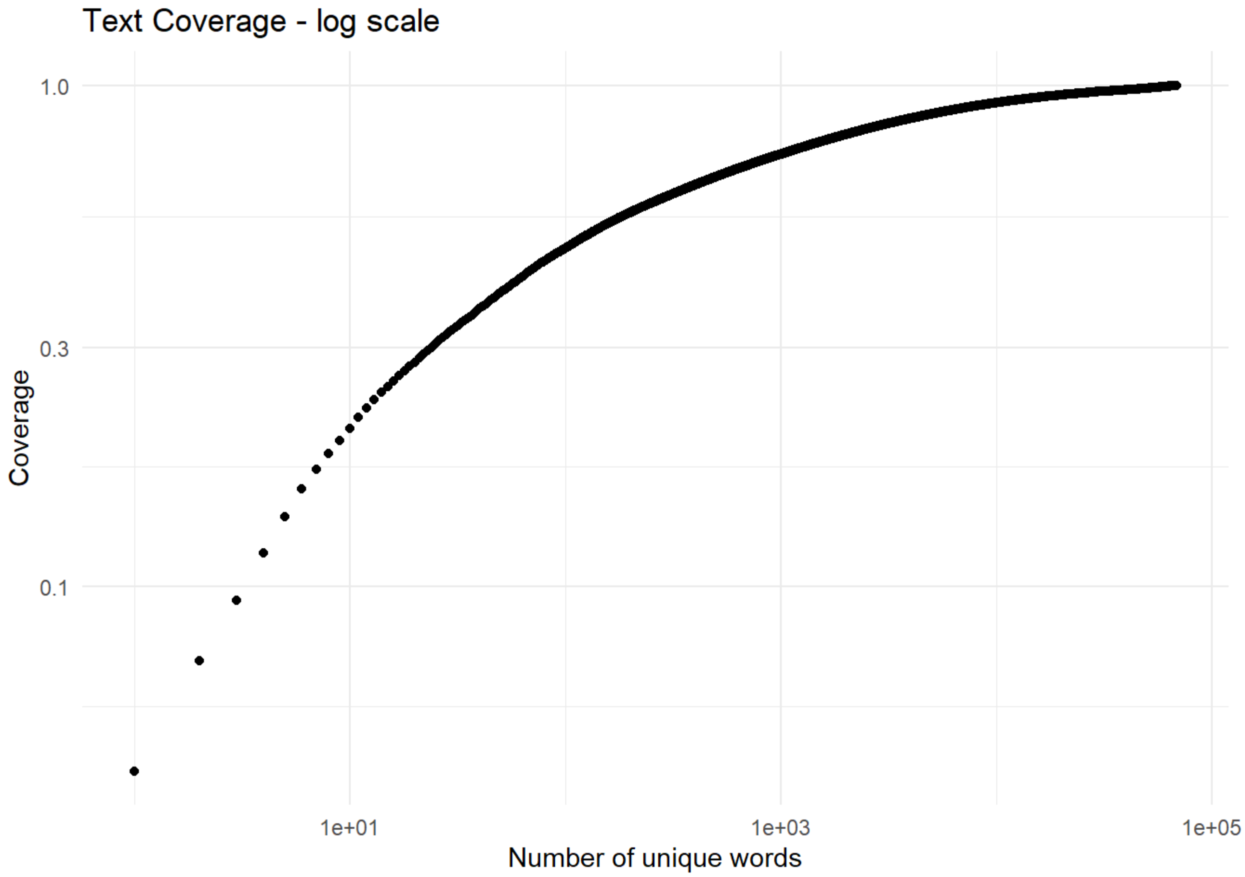
Tokenization

Ngrams

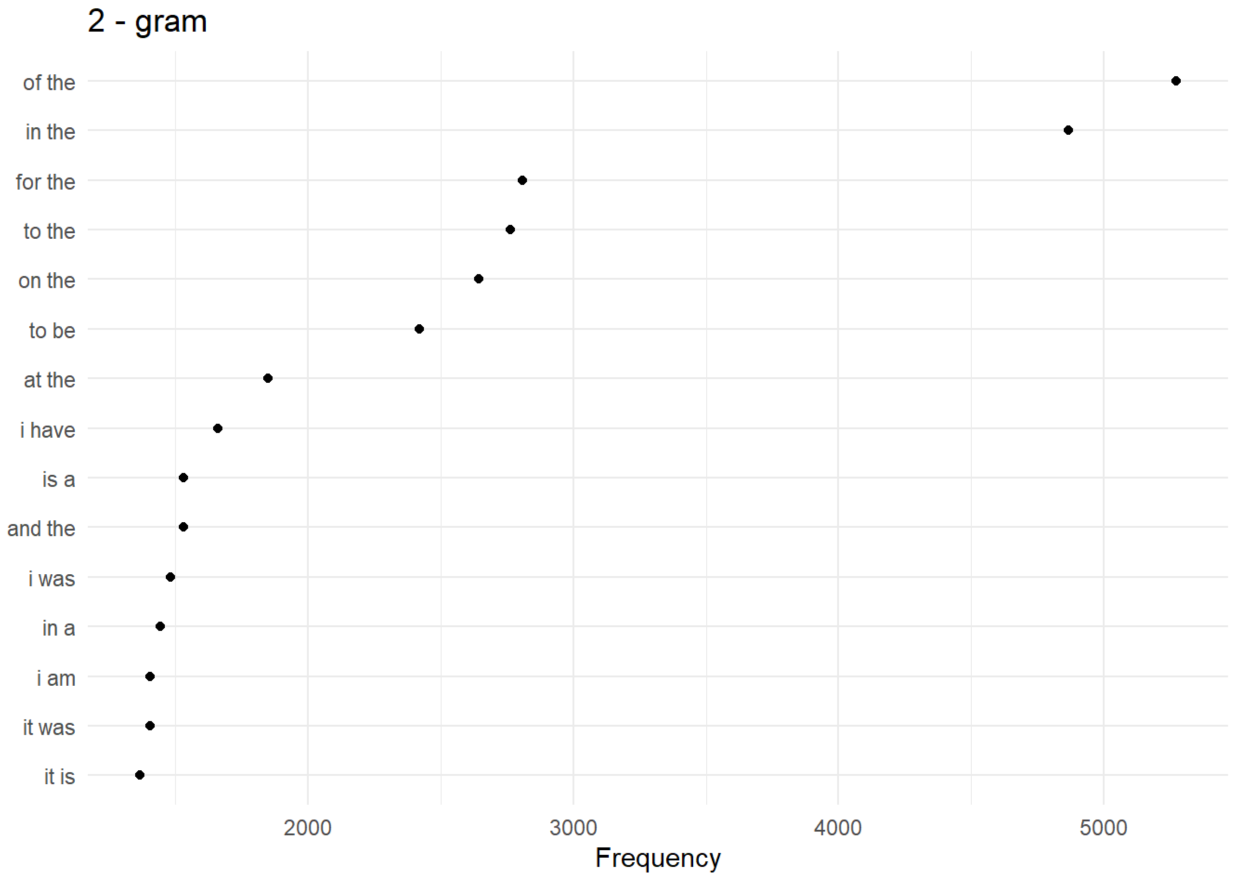
Unigrams

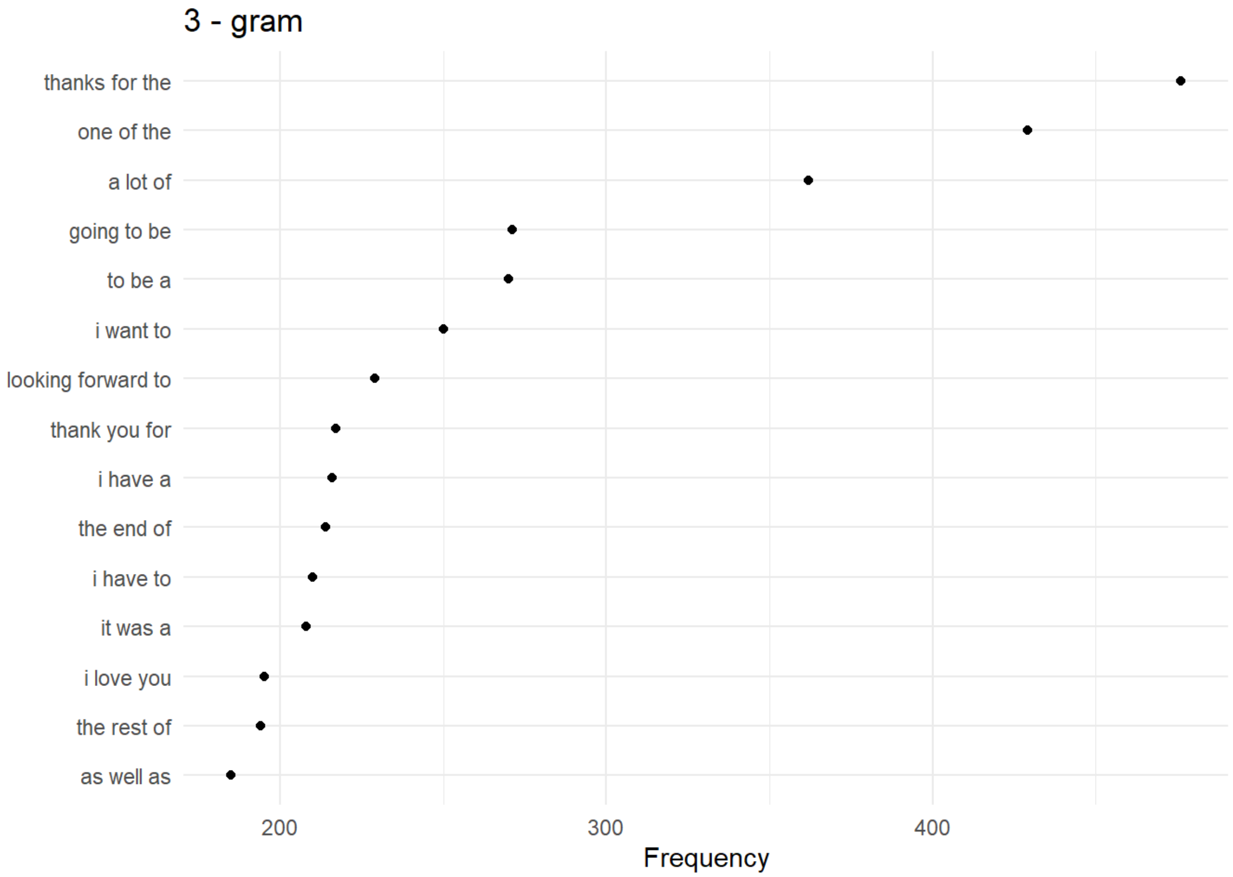

Single word frequency
the
to
and
a
i
of
in
you
is
for
it
that
on
my
with
this
be
was
have
at
are
but
me
we
so
as
not
your
all
just
20000
40000
60000
Frequency

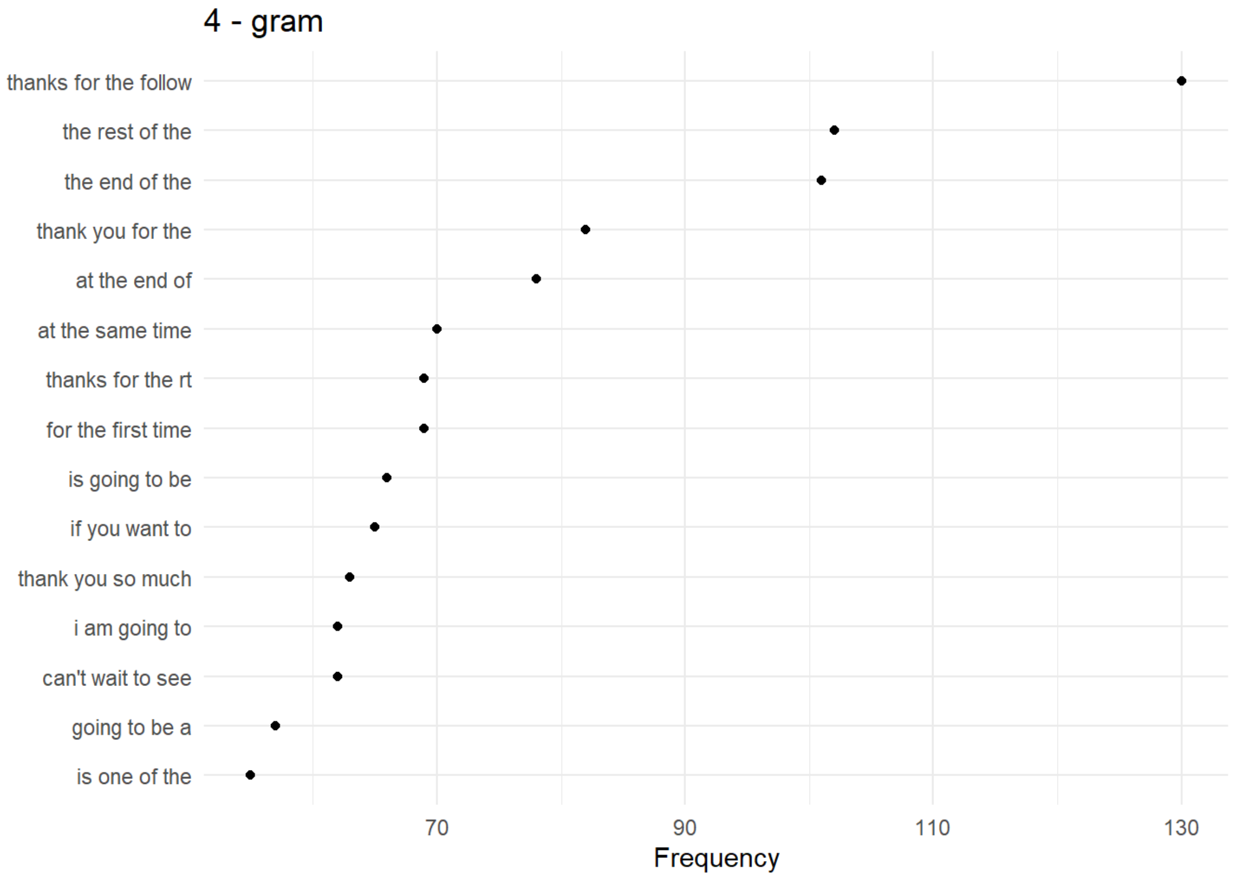

Text Coverage
Coverage
1.00
0.75
0.50
0.25
0.00
0
20000
40000
60000
Number of unique words

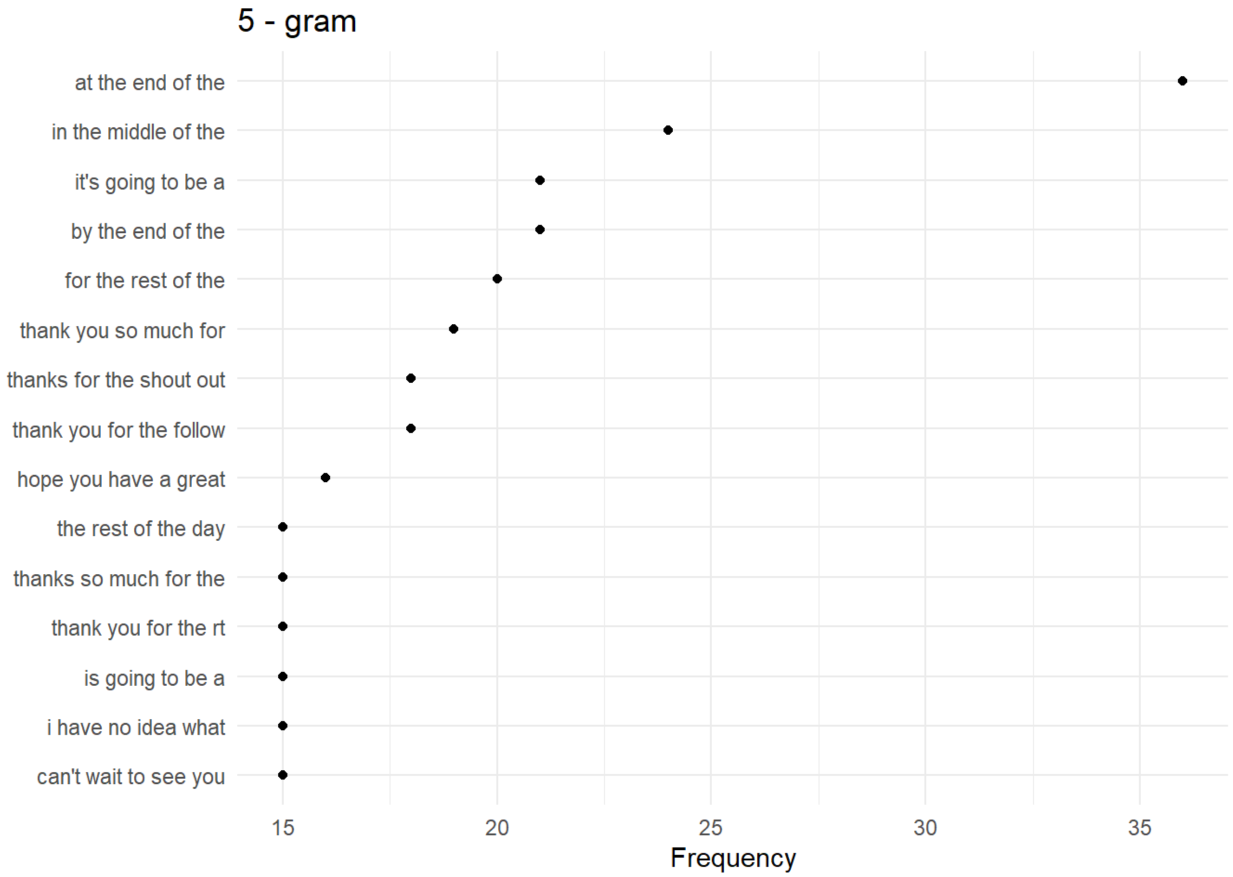


Higher n-grams


2 - gram
of the
in the
for the
to the
on the
to be
at the
i have
is a
and the
i was
in a
i am
it was
it is
2000
3000
4000
5000
Frequency


3 - gram
thanks for the
one of the
a lot of
going to be
to be a
i want to
looking forward to
thank you for
i have a
the end of
i have to
it was a
i love you
the rest of
as well as
200
300
400
Frequency


4 - gram
thanks for the follow
the rest of the
the end of the
thank you for the
at the end of
at the same time
thanks for the rt
for the first time
is going to be
if you want to
thank you so much
i am going to
can't wait to see
going to be a
is one of the
70
90
110
130
Frequency



Findings

Three different text data sources are analyzed in this assignment: blogs, news and twitter. Together these sources represent over 3 million sentences containing in total over 70 million words.

The text data is sampled for efficient data handling. Next the data is cleaned (tokenization, profanity filtering) and further analysis is performed on the frequency of the words.

As one might expect, the most frequent words are often the simplest words, like “the”, “to” and “and”. The coverage of all words in a text is dominated by such high-frequency words. By using only 125 words, 50% of the text can be covered. By using 1000 words already 72% is covered. This insight provides possibilities to create a efficiently small dictionairy of the words which must be included in the model. The frequency of rarely occuring words can be used to predict what to do when unknown words are encountered.

N-grams are used to investigate the frequency of combinations of words. The frequency is strongly dependend on the number of *N*. For bigrams (N=2), the top-combination “of the” is found over 5000 times, whereas for pentagrams (N=5) the top combination “at the end of the” only occurs 36 times. It can be concluded that most focus should be given in a good represenatation of the bigrams and trigrams.

Next steps

Using N-grams, the relative frequency of words can be further analyzed. E.g count all the bigrams which starts with “in”, the count the frequency of the possible second words like “the”, “house”, etc. This gives a first estimation of the suggestion for the predictive model.

Also a weighting between bigrams and trigrams should be incorporated, or antoher form of combination of those two.

Another problem to overcome is to respond to words which are not represented by the training data.

Finally model improvements, like smoothers, should be incorporated in the model.

Appendix: All code for this report

```
knitr::opts_chunk$set(echo = FALSE)
# Load libraries
library(quanteda)
library(stringi)
library(readtext)
require(ggplot2)
require(knitr)
downloadDatafromLink <- function(){
  # Download the data
  # data directory
  datadir <- "./TrainingData"
  if(!file.exists(datadir)){dir.create(datadir)}
  # download zipped data
  zipf = file.path(datadir, "Coursera-SwiftKey.zip")
  if(!file.exists(zipf)){
    fileUrl <- "https://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-SwiftKey.zip"
    download.file(fileUrl,destfile = zipf)
    unzip(zipf,exdir=datadir)
  }
}
# Download the data from the web
downloadDatafromLink()
fp_fromrawdata <- function(text_source, languague_dir){
    # returns total file path to raw data

    # text_source and langauge_dir should be on of the following standard inputs
    # languague_dir = "de_DE" or "en_US" or "fi_FI" or "ru_RU"
    # text_source = "blogs" or "news" or "twitter"
    datadir <- "./TrainingData"
    datadir2 = "final"

    text_source_file = paste(languague_dir, ".", text_source, ".txt", sep="")

    fp_rawdata = file.path(datadir, datadir2, languague_dir, text_source_file)
    fp_rawdata
}
readAllRawData <- function(fp_rawdata){
  # Reads a raw data file completely
    con <- file(fp_rawdata, "r")
    rawData <- readLines(con, skipNul = TRUE)
    close(con) ## It's important to close the connection when you are done.
    rawData
}
getbasicsummary <- function(text_source){
  # Get file path
  fp_raw <- fp_fromrawdata(text_source, "en_US")
  # read raw data
  text_en <- readAllRawData(fp_raw)
  # get basics
  size <- file.size(fp_raw)/1024^2
  linecount <- length(text_en)
  wordcount <- sum(stri_count_words(text_en))
  c(size, linecount, wordcount)
}

createSampleDataEn <- function(text_source, sample_fp){
  # read raw data
  text_en <- readAllRawData(fp_fromrawdata(text_source, "en_US"))
  # Randomly Sample a limited percentage of the data
  set.seed(15-7-2020)
  text_sample <- text_en[rbinom(length(text_en), 1, .02)==1]
  # Write samples to file
  write.table(text_sample, file = sample_fp, row.names = FALSE, col.names=FALSE, sep=",", quote =FALSE)
}

# Read samples, create from raw data if necessary
text_sources = c("blogs","news","twitter")
sampledir <- "./TrainingData_Samples_02perc/"

# df to store basic info
basic_cols <- c("Size [MB]", "No. of lines", "No. of words")
basic_summ = data.frame(matrix(vector(), 3, 3,
                dimnames=list(text_sources, basic_cols)),
                stringsAsFactors=F)
names(basic_summ) <- basic_cols

for (text_source in text_sources){
  # get basics info
  basic_summ[text_source,basic_cols] <- getbasicsummary(text_source)
  # Read samples, create from raw data if necessary
  sample_fp <- paste0(sampledir,text_source, "_sample.txt")
  # If file not exists, create it
  if (!file.exists(sample_fp)) {
    if (!dir.exists(sampledir)) {
      dir.create(sampledir)
    }
    createSampleDataEn(text_source, sample_fp)
  }
}
kable(basic_summ)
# read data samples
texts <- readtext(paste0(sampledir,'*.txt'), docvarsfrom="filenames",dvsep="en_US.",verbosity=1, encoding="UTF-8")
# put texts in corpus
texts_corpus_all <- corpus(texts)
```

```
# reshape to improve performance
texts_corpus <- corpus_reshape(texts_corpus_all, to = "sentences")
# Tokenize data, remove punctuation, sybmbols, numbers and URLS
toks_filtered <- tokens(texts_corpus,
                      remove_punct = T,
                      remove_symbols = T,
                      remove_numbers = T,
                      remove_url = T,
                      ) %>%
               tokens_tolower()

# Profanity filter
# list downloaded from internet: https://github.com/LDNOOBW/List-of-Dirty-Naughty-Obscene-and-Otherwise-Bad-Words/blob/master/en
profanity_list <- read.csv("List-of-Dirty-Naughty-Obscene-and-Otherwise-Bad-Words/en.txt", header=FALSE, stringsAsFactors=FALSE)
toks_filt_clean <- tokens_remove(toks_filtered, pattern = profanity_list)
## single word frequency

# Document feature matrix
dfm_text <- dfm(toks_filt_clean)
# summarize texts
text_sum <- textstat_summary(dfm_text)

# Plot single word frequency
swf_plot <- dfm_text %>%
  textstat_frequency(n = 30) %>%
  ggplot(aes(x = reorder(feature, frequency), y = frequency)) +
    geom_point() +
    coord_flip() +
    labs(x = NULL, y = "Frequency") +
    ggtitle("Single word frequency") +
    theme_minimal()
swf_plot
# compute coverage, including all words
```

```
coverage <- textstat_frequency(dfm_text, n= length(dfm_text))
coverage$index <- 1:dim(coverage)[1]
coverage$relative_frequency <- coverage$frequency / sum(coverage$frequency)
coverage$coverage <- cumsum(coverage$relative_frequency)

# Plot coverage
cvr_plot <- coverage %>%
  ggplot(aes(x = index, y = coverage)) +
  geom_point() +
  labs(x = "Number of unique words", y = "Coverage") +
  ggtitle("Text Coverage") +
  theme_minimal()
cvr_plot
# Plot coverage on log scale
cvr_plot2 <- cvr_plot + scale_x_log10() + scale_y_log10() + ggtitle("Text Coverage - log scale")
cvr_plot2
# plot function
plot_top_features <- function(dfm_, n){
  dfm_ %>%
    textstat_frequency(n = n) %>%
    ggplot(aes(x = reorder(feature, frequency), y = frequency)) +
    geom_point() +
    coord_flip() +
    labs(x = NULL, y = "Frequency") +
    theme_minimal()
}
# plot ngram from n and tokens
ngram_from_tokens <- function(toks, n, excludestopwords){
  # filter stopwords
  if (excludestopwords){
    toks <- tokens_remove(toks, pattern = stopwords('en'))
  }
  # Create ngram
  toks_ngram <- tokens_ngrams(toks, n = n, concatenator = " ")
  # remove duplicate texts like "ugly ugly ugly ugly"
  toks_ngram <- tokens_remove(toks_ngram, pattern="\\b(\\w+)\\s+\\1\\b", valuetype = "regex")
  # Document feature matrix
  dfm_text <- dfm(toks_ngram)
  # Plot
  g <- plot_top_features(dfm_text, 15) + ggtitle(sprintf("%s - gram", n))
}
excludestopwords = FALSE
for (n in c(2,3,4,5)){
  g<- ngram_from_tokens(toks_filt_clean, n, excludestopwords)
  print(g)
}
```