Milestone Report

Igor Tomashevskiy

Friday, March 11, 2016

The report provides the analysis of data from twitter, news and blogs provided for the Capstone project. The purpose of this analysis is to provide a model for text prediction.

Data Transformations:

```
blogs<-readLines("en_US.blogs.txt")
news<-readLines("en_US.news.txt")
tws<-readLines("en_US.twitter.txt")
blogs_line<-length(blogs)
news_line<-length(news)
tws_line<-length(tws)
```

The news data set contains 1010242 lines, the blog data set contains 899288 lines and twitter data set contains 2360148 lines. Word count for each file can be done by using wc function from qdap R package.

```
library(qdap)
```

```
## Loading required package: qdapDictionaries
## Loading required package: qdapRegex
## Loading required package: qdapTools
## Loading required package: RColorBrewer
##
## Attaching package: 'qdap'
##
## The following object is masked from 'package:base':
##
##     Filter
```

```
word.count.blog<-wc(blogs,byrow=FALSE,digit.remove=TRUE)
word.count.news<-wc(news,byrow=FALSE,digit.remove=TRUE)
word.count.tws<-wc(tws,byrow=FALSE,digit.remove=TRUE)
```

word.count.blog = 36893516, word.count.news = 33376745, word.count.tws = 29430648. Please note that digits were removed from the data. The next step is to randomly select records for testing, since the original files are too big. The sample() function will help us select 1% of records from each file:

```
set.seed(12345)
blog_idx<-sample(blogs_line,size=floor(blogs_line/100))
news_idx<-sample(news_line,size=floor(news_line/100))
tws_idx<-sample(tws_line,size=floor(tws_line/100))
```

We use indices to select records from the data:

```
sample.blogs<-blogs[blog_idx]
sample.news<-news[news_idx]
sample.tws<-tws[tws_idx]
sample.data<-c(sample.blogs,sample.news,sample.tws)
writeLines(sample.data,con="sample_data.txt","\n")
```

Now, after setting up all the necessary elements, we store the sample data in a corpus. The corpus is created from text which is stored in the character vector sample.data. The standard for text analysis in R is the tm package. It provides functionality to perform the most common data preparation operations.

```
library(tm)
sample.corpus<-Corpus(VectorSource(sample.data))
```

Next we need to do some data cleansing: remove numbers, remove punctuation, remove stop words, convert all letters to lower case, strip whitespace and perform a stemming of terms.

```
library(stringr)
library(SnowballC)
sample.corpus<-tm_map(sample.corpus,removeNumbers)
sample.corpus<-tm_map(sample.corpus,str_replace_all,pattern="[^[:alnum:]]",replacement=" ")
sample.corpus<-tm_map(sample.corpus,removeWords, words=stopwords("en"))
sample.corpus<-tm_map(sample.corpus,tolower)
sample.corpus<-tm_map(sample.corpus,stripWhitespace)
sample.corpus<-tm_map(sample.corpus,stemDocument)
corpus.clean<-tm_map(sample.corpus,PlainTextDocument)
tdm<-TermDocumentMatrix(corpus.clean)
```

Next step is to construct term-document matrices on n-gram. Within the tm framework, we can construct n-grams using the R interface to the Weka program using the RWeka package.

```
library(RWeka)
OnegramTokenizer<-function(x){NGramTokenizer(x,Weka_control(min=1,max=1))}
tdm_onegram<-TermDocumentMatrix(corpus.clean,control = list(tokenize=OnegramTokenizer))
BigramTokenizer<-function(x){NGramTokenizer(x,Weka_control(min=2,max=2))}
tdm_bigram<-TermDocumentMatrix(corpus.clean,control = list(tokenize=BigramTokenizer))
ThreegramTokenizer<-function(x){NGramTokenizer(x,Weka_control(min=3,max=3))}
tdm_threegram<-TermDocumentMatrix(corpus.clean,control = list(tokenize=ThreegramTokenizer))
```

The disadvantage of a term-document matrix is the fact that the matrix is very big and can exceed R's calculations limits, as a result we will see overflow error message.

```
tdm_onegram
```

```
## <<TermDocumentMatrix (terms: 53449, documents: 42695)>>
## Non-/sparse entries: 533035/2281472020
## Sparsity           : 100%
## Maximal term length: 54
## Weighting          : term frequency (tf)
```

We need to remove sparse terms,with sparsity threshold of 0.99:

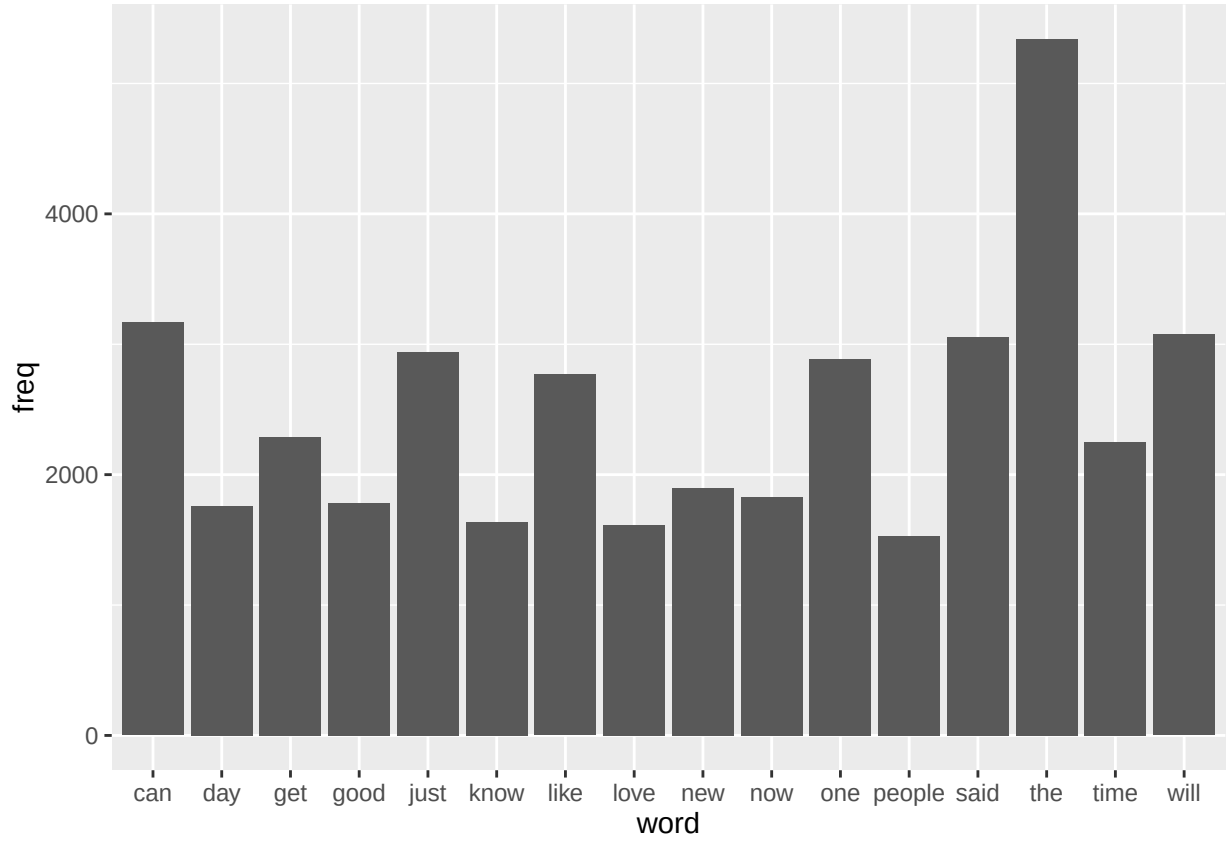
```
tdms_onegram<-removeSparseTerms(tdm_onegram,0.99)
```

Using the tdms_onegram we can find the frequency count of all words in a corpus and plot the frequency of words that used at least 1500 times in the corpus:

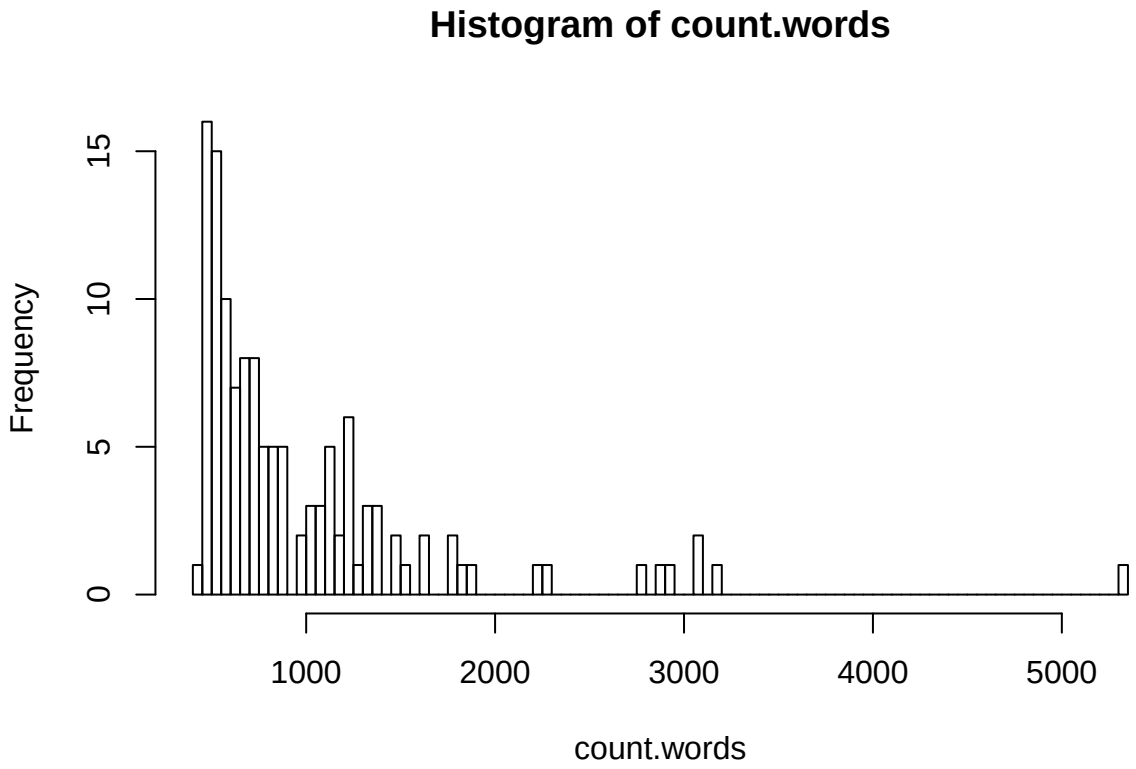
```
library(ggplot2)
```

```
##
## Attaching package: 'ggplot2'
##
## The following object is masked from 'package:NLP':
##
##     annotate
##
## The following object is masked from 'package:qdapRegex':
##
##     %+%
```

```
word.freq<-sort(rowSums(as.matrix(tdms_onegram)),decreasing=TRUE)
word.freq.df<-data.frame(word=names(word.freq),freq=word.freq)
word.freq.df.1500<-subset(word.freq.df,freq>1500)
plot1<-word.freq.df.1500%>%ggplot(aes(word,freq))+geom_bar(stat="identity")
plot1
```



```
count.words<-rowSums(as.matrix(tdms_onegram))
count.words<-as.numeric(count.words)
hist(count.words,breaks=100)
```



Possible Application Design: 1.Create frequency tables for n-gram matrices (n=1,2,3), use them for probability evaluations.
2.Condition the likelihood of "word" in the context of previous words.
3.The probability of the word depends on the probabilty of the n previous words(Markov assumption) .

4. Compute the product of conditional probabilities.
   Questions to consider:

5. For higher n more data is needed.

6. More data will exceed R calculations limits

7. A separate test corpus should be used to evaluate the model.

8.