|  |
| --- |
| Close-up image showing the leaf-sides of two oversized books side-by-side on a bookshelf, with additional books in soft focus background |
| Text Mining  Assignment - 3 |
| |  |  |  | | --- | --- | --- | | Pooja Sheth | 4/15/18 | UW MSBA | |

**Chapter -2**

**Summary of the chapter and the codes**

This chapter does highlight on the understanding of what is text mining and the types of text mining.

* It is always better to have the training sets and subjectivity lexicons, where the sentiments are explicitly defined. As similar texts could be interpreted in different ways, owing to the language and cultural differences.
* Text mining is considered quite like machine learning. It is too based on the analysis on the past data to base new analysis and outcomes.
* It even explains on the steps to be carried out using different packages and functions for analysis the text.
* Various methods of cleaning and structuring data and then coming up with different insights to understand the objective.

Text Mining is defined as the ability to extract useful and fresh insights from large amount of unstructured language, which can be used affectively for decision making. The documents could be csv files or text documents. These collections of files form the corpus. Reading entire corpus manually is not required and not even multiple analysts reading it. There are two types of text mining:

* Bag of words - This technique considers every word or group of words as n grams, as an unique feature of the document. It is easy of organize the corpora of text mining. It is a simple methodology for manipulating the unstructured strings to structure in the Document Term Matrix or Term Document Matrices.
  + DTM TDM matrices are basically used to understand the frequency of specific words in different documents in the corpus. If there are thousands of words in the corpus then it gives good frequency analysis of the words.
* Syntactic Phrase - It differs from bag of words in its complexity. It is based on the syntax of the words. It uses part of speech tagging techniques to identify words in grammatical or useful context. Then blocks of data are created which are then analyzed to create insights out of them.

Before everything a system local is set up. Setting up system location helps to overcome errors associated with unusual characters not recognized by R’s default locale. Post which all the required packages and libraries are installed for the analysis.

For this example, we would be using Bag of words technique. The example is using the twitter data to understand delta airlines customer service tweets. Delta tweets from the twitter API from October 1 to October 15, 2015 is used. The data is cleaned and extracted for the analyses.

* The nchar() is used to understand the length of the social customer service reply. The head() references to the first six rows of the corpus and the last column containing the text. The mean(nchar()) gives the average of the length.
  + The average length of a social customer service reply is 92.37. So, 92 is approximate length. Since the tweet length is 140, the agents are concise and not elaborating on the tweets.
* The sub function looks for pattern match in the string and replaces it> here it replaces thanks to thank you and pls to please and the desired location of change of rows and columns is mentioned too.
* The gsub function is the global substitution function and will replace not just the first instance of the pattern but all the instances. It is even used for removing the specific characters in the entire corpora. Here it substitutes the text mining to tm in the entire document and the &amp to a blank. It could even use to remove all the punctuations in the document.
  + The last parameter of ignore.case = F, this tells the function to explicitly match the pattern. Changing the case from F to T will help to match either upper or lowercase strings.
* If we must make changes in the document for multiple global substitutions. For this the mgsub() can be used from the qdap package. The patterns and replacement would be the inputs and given as inputs to the mgsub(). The mgsub() does multiple substitutions. It substitutes good with great, text mining with tm.
* The paste function is used for concatenation function. Lubridate package is used to switch the newly created dates into an official date format. This date can be used to understand the agent workload.
* The strsplit function creates subset strings by matching character patterns. In this case agent is identified using the string split on the asterisk. As the end two characters post the asterisk recognize the initials.
* If the agent uses some other pattern before their initials, then the agent would be missed out. So the last.chars() can be used to create the numbers for substring dynamically. The last.chars() extracts the last 5 strings in the text and extracts the word great here.
* The subset creates a subset of dates between the given dates and then the table() is used to extract the last two characters of the tweet to know the agent who tweets. The table would give frequency of the number of tweets by the agents.
* The grep and grepl are the functions are used for searching regular expression patterns. The input parameters would be to understand which patterns to search, where to search and whether the lower case and upper case would matter. The grep would return the positions of the tweet that contain the pattern at least once. The grepl returns the logical vector of true or false. Either the words are present or absent.
* If we want to find the number of times the word is present in the string the stringi package is used, which has various functions of stri\_count as well.
* Like grep, if we want to search for patterns using and then the stringr function is used.
* After understanding the patterns the cleaning is performed by removing the stopwords, punctuation, whitespace, numbers, stem document, tolower, removewords. This can be done in the clean corpus. “tm\_map” is an interface function for transforming entire corpora.
* Before the cleaning the tweets are defined as objects of corpus, and in this case it would the unique tweet IDs.
* Once the corpus is cleaned, spell checking is performed. The misspelled words in the tm.definitions are identified from the qdap dictionary. There is a function in qdap that allows to select the correct spellings from the list of possible terms. Apart from this the custom functions can be applied to single string or entire vector of documents.
* Using the tdm as mentioned earlier matrices are created to know the frequency of the words and then rowsums is used to understand the top 10 highly frequent words. These would be Please, sorry, team, confirmation, flight.

Stringi and Stringr are the functions used in R for string manipulation.

this is necessary for nchar to work

```{r}

options( stringsAsFactors = F)

Sys.setlocale('LC\_ALL','C')

```

[1] "C"

```{r setup, include=FALSE}

knitr::opts\_chunk$set(echo = TRUE)

```

read in the file using nchar to determine how many characters i each string

```{r read in the file}

text.df <- read.csv("D:/Lenovo backup/UW/Q3/Text mining/oct\_delta.csv")

x = head(text.df$text)

x

class(x)

nchar(x)

nchar(head(text.df$text))

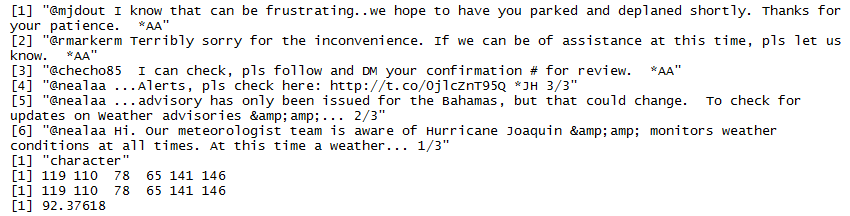
#what is the mean number of characters in the messages

mean(nchar(text.df$text))

#remove blank docs

subset.doc <- subset( text.df, nchar( text.df $ text) > 0)

```



Use string functions to replace terms into one.

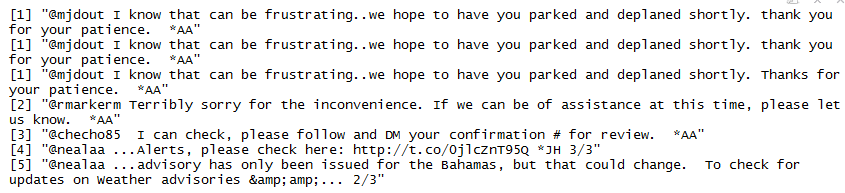
```{r examine using sub}

sub('thanks','thank you', text.df[ 1,5], ignore.case = T)

sub('Thanks','thank you', text.df[ 1,5], ignore.case = T)

sub(' pls',' please', text.df[ 1: 5,5], ignore.case = F)

```



Explore working with string functions

sub vs gsub (gsub changes all occurrences)

```{r exploring string functions}

fake.text<- 'R text mining is good but text mining in python is also'

sub('text mining','tm', fake.text, ignore.case = F)

fake.text<- 'R text mining is good but text mining in python is also'

gsub('text mining','tm', fake.text, ignore.case = F)

#remove the ampersand occurrences for the fifth tweet

text.df[5,5]

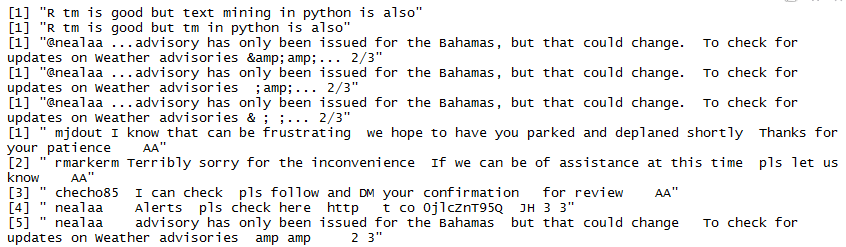
gsub('&amp',' ', text.df[ 5,5])

gsub('amp', ' ', text.df[5,5])

#rmove all punctuation

gsub('[[:punct:]]',' ', text.df[ 1: 5,5])

```



The qdap package provides a lot of useful text mining tools

offers a very convenient wrapper for gsub.

It is called mgsub or multiple global substitutions.

an R programmer can pass a vector of pattern matches to be replaced with another vector.

It is compact and makes repeating many multiple substitutions easy.

To begin create a string vector of patterns to be matched.

Then create another string vector of replacements.

Lastly invoke the mgsub function applied to the fake.text object

In the code below âœgoodâ will be replaced with âœgreat,â âœalsoâ will be replaced with âœjust as suitable,â and âœtext miningâ will be replaced by âœtm.â

```{r use qdap library to learn about patterns strings}

library( qdap)

patterns <- c('good','also','text mining')

replacements <- c('great','just as suitable','tm')

mgsub( patterns, replacements, fake.text)

```

[1] "R tm is great but tm in python is just as suitable"

Another function for dealing with multiple columns is the paste function

In this example replace the abbreviation by the numbers for months

```{r use paste function}

patterns = c('Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec')

replacements = seq( 1: 12)

text.df$month = mgsub(patterns, replacements, text.df$month)

text.df$combined = paste(text.df$month, text.df$date, text.df$year, sep ='â“')

```

Work with strings of text

Convert dates into date format - this will enable examining time series analysis

Separate out identifying information such as agen's initials

Split the strings to get seleted parts

```{r prepare for date functions}

library( lubridate)

text.df $ combined <- mdy( text.df $ combined)

text.df$combined[1:2]

#review the text for first two tweets

text.df $ text[ 1: 2]

# String split on the asterisk to identify the agent for each tweet.

agents <-strsplit( text.df $ text,'[\*]')

agents[1:2]

#develop a function to get selected parts

#start with example - get the last five characters starting at location 18 and ending at location 22

substring('R text mining is great', 18,22)

# the lastchars function

last.chars <- function(text, num){

last <- substr(text, nchar(text)- num + 1, nchar( text))

return(last)

}

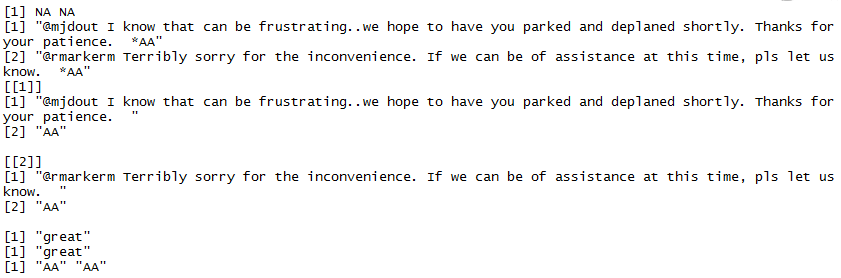
#same code as above example

last.chars('R text mining is great', 5)

#now the tweets

last.chars( text.df $ text[ 1: 2], 2)

```



now work with the tweets to collate the number of tweets by agent

this will involve first referencing tweets over a date range

Note that some agents are miscoded

```{r count tweets by agent}

#first subset the tweets that occurred from oct 5 to oct 9

weekdays <- subset( text.df, text.df$combined >= mdy('10â“05â“2015') &

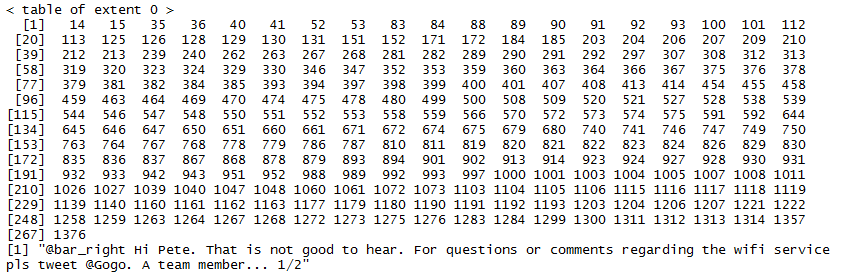
text.df $ combined <= mdy('10â“09â“2015'))

table( as.factor( last.chars( weekdays $ text, 2)))

grep('/2', text.df $ text, ignore.case = T)

text.df$text[15]

```



Search for specific patterns - sorry in this case and keep track of which tweets contained sorry

Grep function look for at least one occurrence

Stringi counts the number of times a pattern occurs in a document

```{r search for specific patterns}

# use grep to search for text

grep('sorry', text.df $ text, ignore.case = T)

#look at one of the tweets

text.df$text[354]

#assign th3 occurrence or not to a vector - it will contain values of TRUE or FALSE

sorry<-grepl('sorry', text.df $ text, ignore.case = T)

#count how many documents 'sorry' showed up and divide the number of documenets

sum( sorry)/ nrow( text.df)

#what about determining how many are apologetic as in sorry or apologize

grep( c('sorry | apologize'), text.df $ text, ignore.case = T)

#check for how many have links

sum( grepl('http', text.df $ text, ignore.case = T))/ nrow( text.df)

#how many have posted a phone number - look for a pattern of 3 numbers or 4 numbers in a row

sum( grepl('[0-9]{3})|[0-9]{4}', text.df $ text))/ nrow( text.df)

#count how many times http occurs in a document

library( stringi)

stri\_count( text.df $ text, fixed ='http')

#

library( stringr)

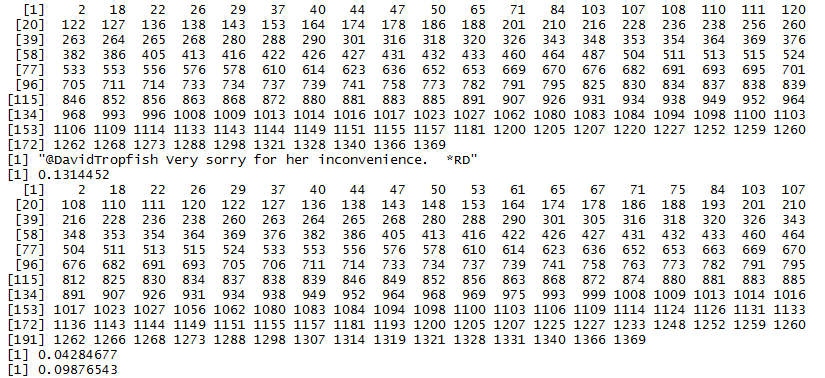
str\_detect( text.df $ text,'http')

#look for the AND of two patterns

patterns <- with(text.df, str\_detect( text.df $ text, 'http') & str\_detect( text.df $ text, 'DM'))

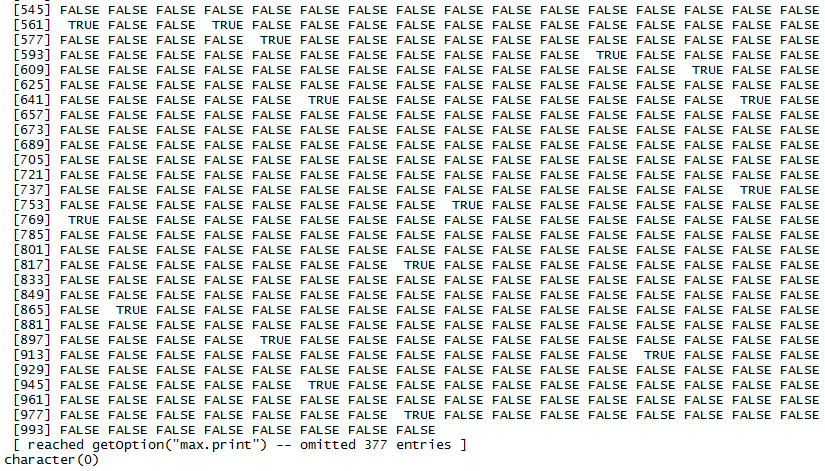
text.df[patterns, 5]

```









text preprocessing and cleaning up

```{r preprocessing and tm actions}

options( stringsAsFactors = FALSE)

Sys.setlocale('LC\_ALL','C')

library( tm)

library( stringi)

#assign an id field

# deprecated tweets <- data.frame( ID = seq( 1: nrow( text.df)), text = text.df $ text)

tweets=data.frame(doc\_id=seq(1:nrow(text.df)),text=text.df$text)

# Return NA instead of tolower error

tryTolower <- function( x){

# return NA when there is an error

y = NA

# tryCatch error

try\_error = tryCatch( tolower( x), error = function( e) e)

# if not an error

if (!inherits( try\_error, 'error'))

y = tolower( x)

return( y)

}

#add to stop words list

custom.stopwords <- c( stopwords('english'), 'lol', 'smh', 'delta')

#clean corpus function using our content\_transformer function

clean.corpus <- function( corpus){

corpus <- tm\_map( corpus, content\_transformer( tryTolower))

corpus <- tm\_map( corpus, removeWords, custom.stopwords)

corpus <- tm\_map( corpus, removePunctuation)

corpus <- tm\_map( corpus, stripWhitespace)

corpus <- tm\_map( corpus, removeNumbers)

return( corpus)

}

#create the corpus from the tweets

corpus <- VCorpus( DataframeSource( tweets))

corpus <- clean.corpus( corpus)

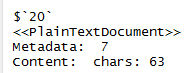
#if you encounter a core issue problem change the tm\_map functions

#tm\_map( corpus, removeNumbers, mc.cores = 1)

#look at the corpus object

as.list(corpus)[20]

```



Work with spell check

Start with some example text

```{r Spell check}

library(qdap)

tm.definition <-'Txt mining is the process of distilling actionable insyghts from text.'

which\_misspelled( tm.definition)

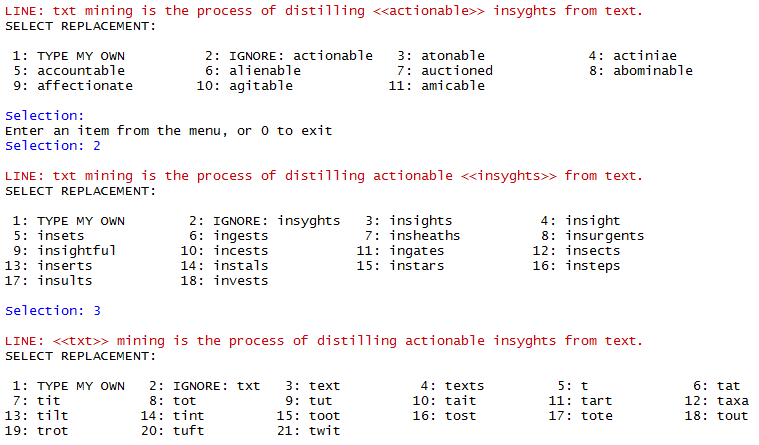
1 8 9

"txt" "actionable" "insyghts"

#the following code runs in interactive mode - you want to use this with caution otherwise you can end up

#with a lot of entry or us escape key

check\_spelling\_interactive( tm.definition)



#create custom spell checker

#fixit uses the first word suggested by qdap which can be problematic as in the example below

fix.text <- function( myStr) {

check <- check\_spelling( myStr)

splitted <- strsplit( myStr, split =' ')

for (i in 1: length( check $ row)) {

splitted[[ check $ row[ i]]][ as.numeric( check $ word.no[ i])] = check $ suggestion[ i]

}

df <- unlist( lapply( splitted, function( x) paste( x, collapse =

' ')))

return (df)

}

fix.text( tm.definition)

```

[1] "text mining is the process of distilling atonable insights from text."

next use the tdm function to create the term document matrix

```{r work with tdm}

tdm <- TermDocumentMatrix( corpus, control = list( weighting = weightTf))

tdm.tweets.m <- as.matrix( tdm)

dim(tdm.tweets.m)

tdm.tweets.m[ 2250: 2255,1340: 1342]

# find frequent terms

term.freq <- rowSums( tdm.tweets.m)

#create a data frame with terms and frequency next to each other

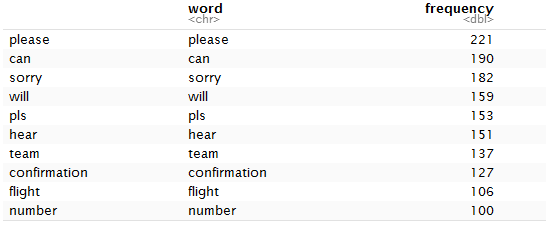
freq.df <- data.frame( word = names( term.freq), frequency = term.freq)

#sort from most to least frequent

freq.df <- freq.df[ order( freq.df[, 2], decreasing = T),]

freq.df[ 1: 10,]

```



**Chapter -3**

**Summary of the chapter and the codes**

The frequency count of the words in the text corpus is an insightful endeavor. Post which the specific association at words can be looked at. This word association is like the correlation analysis in statistics. Qdap package is used to explore this association. Apart from word association, word network can also be used to get the insights. A relationship between words is captured in a special matrix called adjacency matrix. This would explore multiple word linkages simultaneously. And weights would be assigned to various word linkages.

The bar plot could be used for a good representation for the frequency of words. The ggplot2 package is used to sketch the bar plots. And the gg themes and the geom\_text is used to represent the white numerical labels at the end of each bar.

Before plotting the frequency, the cleaning of the corpus is what should be done. And the TDM matrices are used to calculate the frequency.

After the frequency analysis the association analysis is done. The code explores the association with the term apologies. It is important to do the associations of the terms that could deliver insightful results.

The code creates a data frame of factors for each term and the corresponding association values. Once there is a association frame of highly correlated words and their corresponding values, the graph using ggplot2 can be build along with their association values.

R has interpreted the word you as youb, and sometimes these weird parsing does happen which can be removed in the word cloud.

One other way of interpreting the word association is using the word network. Network structures are interesting in conveying multiple types of information visually. Word network indicated the word association under what circumstances. They can be used to understand the word choice by visually producing clusters in the layout.

In the network graph, the lines connecting the circle are called edges and the circles are called nodes or vertices. The size of the node says the prominent members in the network and the thickness says about the strength of the nodes. It even tells which words are associated with which ones, as those would be connected via edges.

The igraph library is used to build the word network graph. The code used the grep and specific pattern match to index the entire data. In this case the pattern refund was chosen based on frequent terms and association analysis performed earlier. Removing the sparse and infrequent terms is another way of reducing the size of the TDM matrix.

Then a small Term document matrix is used to build an example word network.

A small word network of the frequent terms is built from the refund pattern. This is plot using the igraph function. There are 3 networks based on the 3 tweets and the 2 of them are connected using the refund and apologies word.

The qdap package provides a convenient wrapper to create this type of visual. Using qdaps’ word\_netwrok\_plot and word\_associate functions saves considerable time and function. And creates the same visual.

The word association functions network is cluttered compared to the earlier word network as it used all the seven tweets compared to the only three tweets used prior.

Hierarchical dendrograms are a relatively easy approach for word clustering. A dendrogram is a tree like visualization based on the frequency. A reduced term DTM is expressed as the dendrogram for the delta assist corpus. There are clusters formed in the dendrogram.

There is a cluster for the baggage service, so it looks like there are tweets not only for delays but also for baggage service.

A modified dendrogram is created using custom visualizations. The dendrogram confirms the agent behavior asking for customers to follow and direct message the team with confirmation numbers.

Instead of calling the base plot function, the circlize\_dendrogram is used to create a novel illustration. The circular dendrogram highlights the agent’s behavioral insights.

A word cloud is an interesting visualization of the interesting words based on the frequency. There are 3 types of word clouds that can be constructed. The first one is the simple one based on the frequency. The commonality cloud is used to understand the common words used in 2 different corpora.

The comparison cloud is on the contrary used to identify dissimilar words among two or more corpora.

The first world cloud is a simple one 100 words and two colors based on delta’s tweets.

The second one is the commonality word cloud for the words in common in the Amazon and Delta customer service tweets. But the problem is the words are shown only if they are common among the corpora. It does not demonstrate the explicit differences. If one corpus mentions a term once but the other mentions it 100 times, still it would be shown. But the significant difference between the term frequencies is insightful to note.

The pyramid plot is used for this reason, as it would show the shared words and their corpora differences. The pyramid plot unlike commonality cloud can only be constructed with 2 corpora.

The plotrix package is used to create the polarized tag plot, showing words in common between different corpora with the bars indicating the frequency of the words. Here it is used to compare the words in common between amazon and delta. The pyramid.plot is used to edit the aesthetics in the plot, with respect to the words length, labels, colors.

```{r setup, include=FALSE}

knitr::opts\_chunk$set(echo = TRUE)

```

set up the libraries

```{r set up the libraries}

library(SnowballC)

library(tm)

library(ggplot2)

library(ggthemes)

```

read in the text as a single file

```{r read in the text}

#read in the text file from the directory as a csv file

text.df <- read.csv("D:/Lenovo backup/UW/Q3/Text mining/oct\_delta.csv")

#deprecated code from boook - tweets =data.frame( ID = seq( 1: nrow( text.df)), text = text.df $ text)

#the first two columns are defined

#ifusing `DataframeSource` the \*first\* column \*\*MUST\*\* be named `doc\_id` followed by a `text` column.

# Any other columns are considered metadata associated row-wise.

tweets=data.frame(doc\_id=seq(1:nrow(text.df)),text=text.df$text)

```

Now start processing the text

```{r}

#create a function to change case to lower

tryTolower <- function( x){

y = NA

try\_error = tryCatch( tolower( x), error = function( e) e)

if (! inherits( try\_error, 'error'))

y = tolower( x)

return( y) }

```

Extend the stopwords file with additional unwanted

```{r add to stopwords}

#add to stopwords

custom.stopwords = c( stopwords("english"), "lol", "smh", "delta", "amp")

```

make a function that can be used to clean up a corpus

```{r function for cleaning up}

#now set up a function to clean up a corpus

clean.corpus <- function( corpus){

corpus <- tm\_map( corpus, content\_transformer( tryTolower))

corpus = tm\_map( corpus, removeWords, custom.stopwords)

corpus = tm\_map( corpus, removePunctuation)

corpus = tm\_map( corpus, stripWhitespace)

corpus = tm\_map( corpus, removeNumbers)

return( corpus)

}

```

get ready for identifying term frequency from corpus

```{r generater corpus}

#corpus <â“ VCorpus( DataframeSource( tweets), readerControl = list( reader = meta.data.reader))

#create the frequency dataframe

corpus <- VCorpus( DataframeSource( tweets))

corpus <- clean.corpus( corpus)

tdm <- TermDocumentMatrix( corpus, control = list( weighting = weightTf))

tdm.tweets.m <- as.matrix( tdm)

term.freq <- rowSums( tdm.tweets.m)

freq.df <- data.frame( word = names( term.freq), frequency = term.freq)

freq.df <- freq.df[ order( freq.df[, 2], decreasing = T),]

```

use ggplot to plot the most frequent

```{r plot the term frequency}

# plot the terms by frequency

freq.df$ word <- factor( freq.df $ word,

levels = unique( as.character( freq.df $ word)))

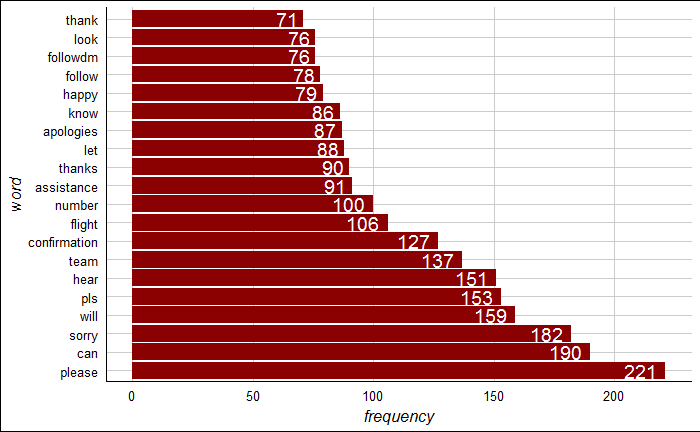
ggplot( freq.df[ 1: 20,], aes( x = word, y = frequency)) +

geom\_bar( stat ="identity", fill ='darkred') +

coord\_flip() + theme\_gdocs() +

geom\_text( aes( label = frequency), colour ="white", hjust = 1.25, size = 5.0)

```



Find the associations for a given term

and convert to a data fram for plotting

```{r get associations}

#find associations

associations = findAssocs( tdm, 'apologies', 0.11)

associations = as.data.frame( associations)

associations $ terms = row.names( associations)

associations $ terms <- factor( associations $ terms, levels = associations $ terms)

```

Plot the associations

```{r plot the associations}

#plot the associations

ggplot( associations, aes( y = terms)) +

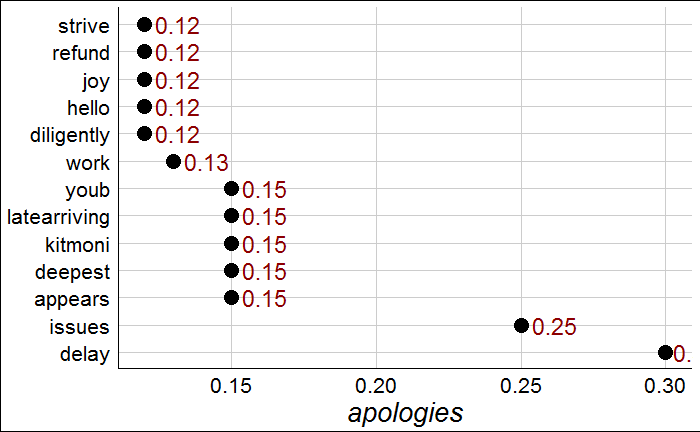
geom\_point( aes( x = apologies), data = associations, size = 5) +

theme\_gdocs() +

geom\_text( aes( x = apologies, label = apologies), colour ="darkred", hjust = -0.25, size = 6) +

theme( text = element\_text( size = 20), axis.title.y = element\_blank())

```



Create word networks based on the associations

firs find tweets with a targeted key word

```{r return tweets with a key word}

library( igraph)

refund <- tweets[ grep("refund", tweets$text, ignore.case = T), ]

```

Get the corpus based on the targeted key word

```{r corpus based on key word}

#refund.corpus <â“ VCorpus( DataframeSource( refund[ 1: 3,]), readerControl = list( reader = refund.reader))

refund.corpus <-VCorpus( DataframeSource( refund[1:3,]))

refund.corpus <- clean.corpus( refund.corpus)

refund.tdm <- TermDocumentMatrix( refund.corpus, control = list( weighting = weightTf))

```

Generate the adjancey matrix for the key terms

This will be based on matrix multiplication

```{r derive the adjancey matrix}

#adj.m <- all %\*% t( all)

library( igraph)

refund.m <- as.matrix( refund.tdm)

refund.adj = refund.m %\*% t( refund.m)

refund.adj = graph.adjacency( refund.adj, weighted = TRUE, mode ="undirected", diag = T)

refund.adj = simplify( refund.adj)

```

Plot the network based on the selected word

this will involve specifying the edge characteristics

```{r plot the network based on the adjancey matrix}

plot.igraph( refund.adj, vertex.shape ="none",

vertex.label.font = 2,

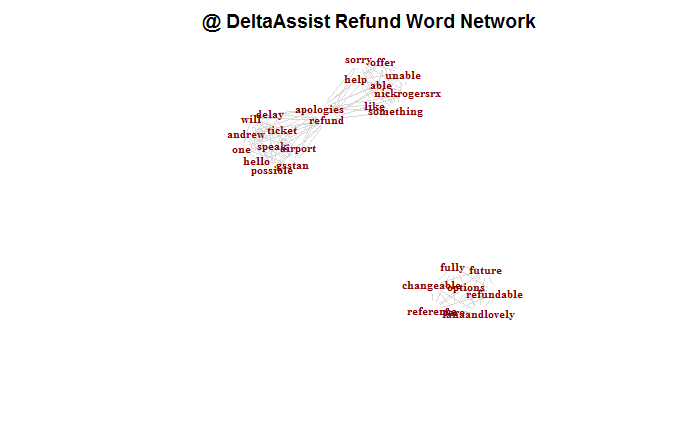
vertex.label.color ="darkred",

vertex.label.cex = .7,

edge.color ="gray85")

title( main ='@ DeltaAssist Refund Word Network')

```



Generate a network with qdap library

This does not require the creation of the tdm or the adjancey matrix

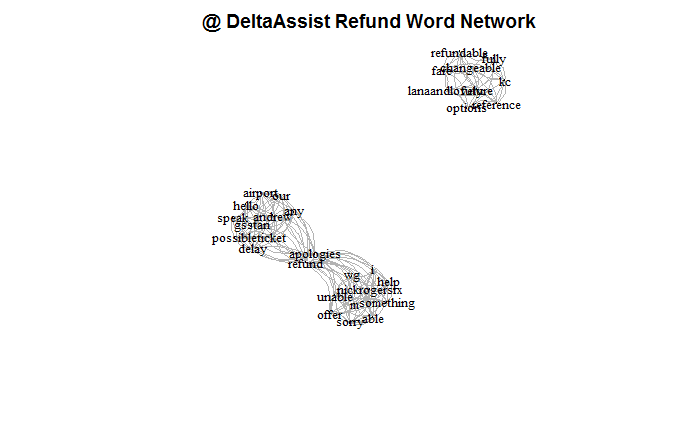
```{r use dqap to create the network}

library( qdap)

word\_network\_plot( refund $ text[ 1: 3])

title( main ='@ DeltaAssist Refund Word Network')

```

Using qdap find tweets with a given word and internally create the adjancey matrix

add more terms by adding a comma and term in quotes

```{r create the network without the extra steps}

word\_associate( tweets $ text, match.string = c('refund'),

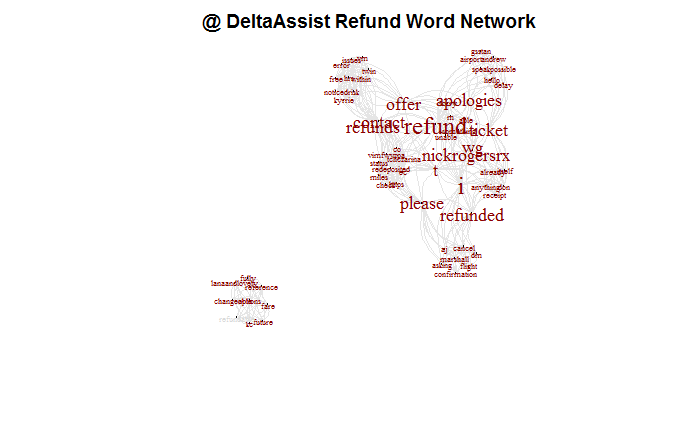
stopwords = Top200Words,

network.plot = T,

cloud.colors = c('gray85','darkred'))

title( main ='@ DeltaAssist Refund Word Network')

```



reduce the number of terms in the matrix based on Sparse terms

and generate a dendogram

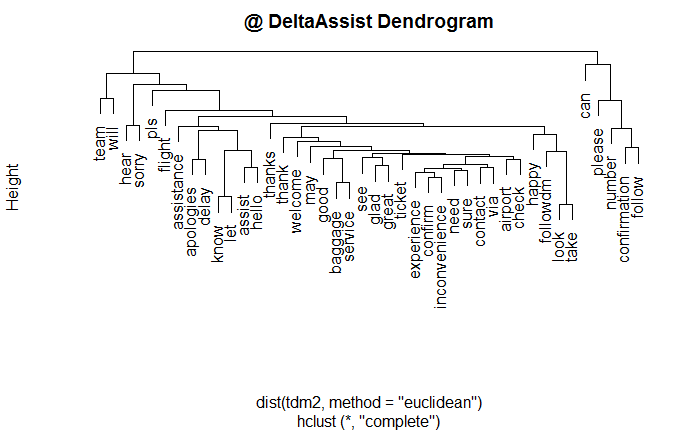
```{r remove sparse terms and plot}

tdm2 <- removeSparseTerms( tdm, sparse = 0.975)

hc <- hclust( dist( tdm2, method ="euclidean"), method ="complete")

plot( hc, yaxt ='n', main ='@ DeltaAssist Dendrogram')

```



add color to the dendrogram

and separate out by different clusters

```{r color the dendrogram}

dend.change <- function( n) { if (is.leaf( n)) {

a <- attributes( n)

labCol <- labelColors[ clusMember[ which( names( clusMember) == a $ label)]]

attr( n, "nodePar") <- c( a $ nodePar, lab.col = labCol)

}

n

}

hcd = as.dendrogram( hc)

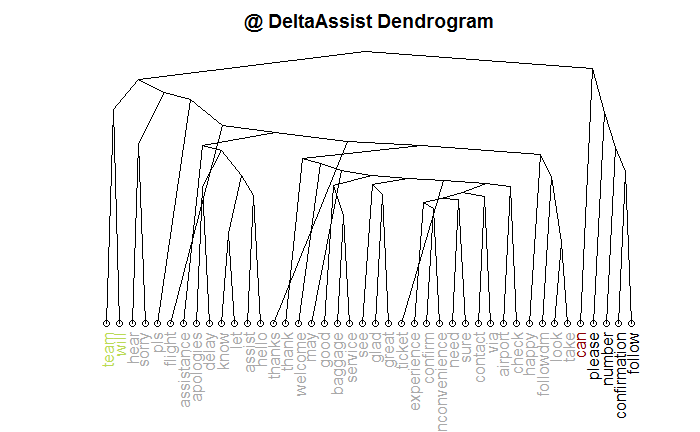
clusMember =cutree( hc, 4)

labelColors = c('darkgrey', 'darkred', 'black', '#bada55')

clusDendro = dendrapply( hcd, dend.change)

plot( clusDendro, main = "@ DeltaAssist Dendrogram", type = "triangle", yaxt ='n')

```



View the dendrogram in a different form

```{r different visualization}

library( dendextend)

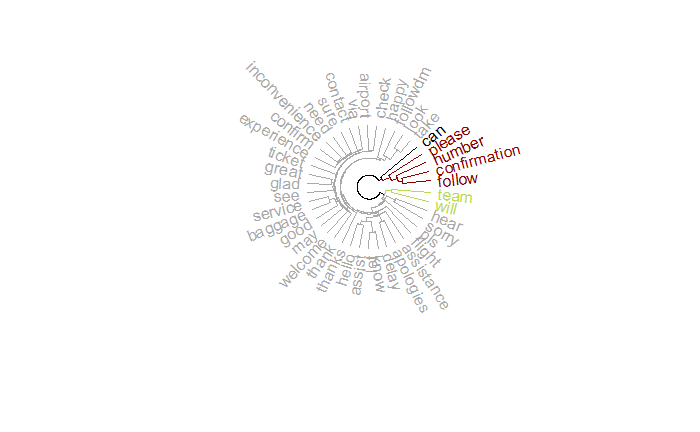
library(circlize)

hcd <- color\_labels( hcd, 4, col = c('#bada55','darkgrey', "black", 'darkred'))

hcd <- color\_branches( hcd, 4, col = c('#bada55','darkgrey', "black", 'darkred'))

circlize\_dendrogram( hcd, labels\_track\_height = 0.5, dend\_track\_height = 0.4)

```



Now we want to see how this information shows up in a word cloud

first set up two functions to clean up the corpus

```{r set up for word cloud view}

library( tm)

library( wordcloud)

tryTolower <- function( x){

y = NA

try\_error = tryCatch( tolower( x), error = function( e) e)

if (!inherits( try\_error, 'error'))

y = tolower( x)

return( y)

}

custom.stopwords <- c( stopwords('english'), 'sorry', 'amp', 'delta', 'amazon')

clean.vec <- function( text.vec){ text.vec <- tryTolower( text.vec)

text.vec <- removeWords( text.vec, custom.stopwords)

text.vec <- removePunctuation( text.vec)

text.vec <- stripWhitespace( text.vec)

text.vec <- removeNumbers( text.vec)

return( text.vec)

}

```

We are going to compare two corpuses

```{r read in the corpuses}

amzn <- read.csv("D:/Lenovo backup/UW/Q3/Text mining/amzn\_cs.csv")

delta <- read.csv("D:/Lenovo backup/UW/Q3/Text mining/oct\_delta.csv")

amzn.vec <- clean.vec( amzn $ text)

delta.vec <- clean.vec( delta $ text)

```

Now collapse both corpus into documents

the purpose is to examine both and compare them

```{r collapse the corpus}

amzn.vec <- paste( amzn.vec, collapse = " ")

delta.vec <- paste( delta.vec, collapse = " ")

all <- c( amzn.vec, delta.vec)

corpus <- VCorpus( VectorSource( all))

```

create a tdm based on the revised corpus

```{r generate the tdm}

tdm = TermDocumentMatrix( corpus)

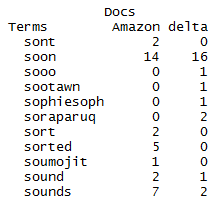
tdm.m = as.matrix( tdm)

#name the columns

colnames( tdm.m)<- c("Amazon", "delta")

tdm.m[3480:3490,]

```



now show the word cloud by calling the commonality plot - shows words that are common to both

```{r generate a commonality cloud}

#show the color palette

display.brewer.all()

#pick purples can be any color

pal <- brewer.pal( 8, "Purples")

#use the darker colors

pal <- pal[-( 1: 4)]

#generate the commonality cloud

commonality.cloud( tdm.m, max.words = 200, random.order = FALSE, colors = pal)

```



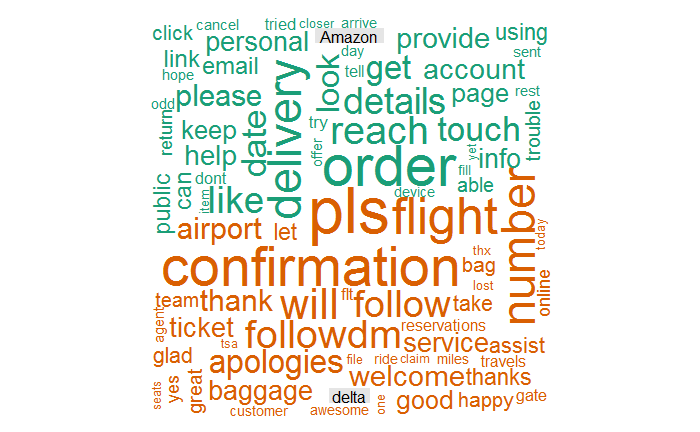
compare the two corpora in the a cloud using different colors

```{r}

comparison.cloud( tdm.m, max.words = 200, random.order = FALSE, title.size = 1.0,

colors = brewer.pal( ncol(tdm.m),"Dark2"))

```



Now let's look at the relative differences between common words by the two corpus

```{r}

library( plotrix)

common.words <- subset( tdm.m, tdm.m[, 1] > 0 & tdm.m[, 2] > 0)

tail( common.words)

#calculate the differences between the two columns of common words

difference <- abs( common.words[, 1] - common.words[, 2])

#combine the differences with the common words

common.words <- cbind( common.words, difference)

#sort by the difference column in decreasing order

common.words <- common.words[ order( common.words[, 3], decreasing = TRUE), ]

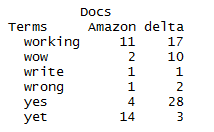
#select the top 25 words and create a data frame

top25.df <- data.frame( x = common.words[ 1: 25, 1],

y = common.words[ 1: 25, 2],

labels = rownames(common.words[ 1: 25, ]))

```



Create a pyramid plot

```{r}

pyramid.plot(top25.df$x, top25.df$y,

labels = top25.df$labels,

#change gap to show longer words

gap = 20,

top.labels = c("Amazon", "Words", "delta"),

main = "Words in Common",

laxlab = NULL, raxlab = NULL, unit = NULL)

```

