Contents

[Data Preprocessing 1](#_Toc449031514)

[What we Learned about Data Science 1](#_Toc449031515)

[Final Thoughts on the Unknown Mushrooms 2](#_Toc449031516)

Introduction

For project number 3, we used several Text Mining libraries in R to try to learn information about the **acq** corpus in the **tm** (text mining) package. As the first part of this project, we tried to fully understand the entire 50 documents of the corpus. To do this, we originally did some initial cleaning of the corpus, ran functions on it to find out more information, re-cleaned the corpus, and then re-ran those functions. We then delved into the top 10 documents, which will be discussed in section 3, that provided more insight into how we can clean the corpus to re-examine all 50 documents. Therefore, we detail in section 4 how we went back and re-looked at the 50 documents to see if any new information. We conclude this project with some final thoughts about the Corpus as well as our thoughts on Data Science from this project.

1. Pre-Processing
   1. Loading Packages

As stated in the introduction, our first goal in this project was to try to find out more information about all of the documents in the entire **acq** corpus. To do this, we first had to install the necessary text mining R packages. After doing some research on the CRAN repository, we found that CRAN has a [Natural Language Processing (NLP) task view](https://cran.r-project.org/web/views/NaturalLanguageProcessing.html) in which we can run the code below that will download and install all of the main NLP or text mining packages:

Load package and data require(ctv)  
install.views('NaturalLanguageProcessing')  
update.views('NaturalLanguageProcessing')

\*Note: we have the update.views function here because some packages may have already been downloaded or there was an issue downloading one so running the update.views will re-download the newer verison.

Additionally, we loaded up several other packages. The full list of packages that were used and tested during this project are below:

require(RColorBrewer)

## Loading required package: RColorBrewer

require(ggplot2)

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 3.2.3

require(tm)

## Loading required package: tm

## Warning: package 'tm' was built under R version 3.2.3

## Loading required package: NLP

## Warning: package 'NLP' was built under R version 3.2.3

##   
## Attaching package: 'NLP'  
##   
## The following object is masked from 'package:ggplot2':  
##   
## annotate

require(SnowballC)

## Loading required package: SnowballC

## Warning: package 'SnowballC' was built under R version 3.2.3

require(textreuse)

## Loading required package: textreuse

## Warning: package 'textreuse' was built under R version 3.2.4

require(wordnet)

## Loading required package: wordnet

## Warning: package 'wordnet' was built under R version 3.2.3

## Warning in initDict(): cannot find WordNet 'dict' directory: please set the  
## environment variable WNHOME to its parent

require(zipfR)

## Loading required package: zipfR

## Warning: package 'zipfR' was built under R version 3.2.3

require(wordcloud)

## Loading required package: wordcloud

## Warning: package 'wordcloud' was built under R version 3.2.4

require(openNLP)

## Loading required package: openNLP

## Warning: package 'openNLP' was built under R version 3.2.3

require(proxy)

## Loading required package: proxy

## Warning: package 'proxy' was built under R version 3.2.4

##   
## Attaching package: 'proxy'  
##   
## The following objects are masked from 'package:stats':  
##   
## as.dist, dist  
##   
## The following object is masked from 'package:base':  
##   
## as.matrix

require(stringr)

## Loading required package: stringr

require(data.table)

## Loading required package: data.table

require(openNLP)  
require(qdap)

## Loading required package: qdap

## Warning: package 'qdap' was built under R version 3.2.4

## Loading required package: qdapDictionaries

## Warning: package 'qdapDictionaries' was built under R version 3.2.3

## Loading required package: qdapRegex

## Warning: package 'qdapRegex' was built under R version 3.2.4

##   
## Attaching package: 'qdapRegex'  
##   
## The following object is masked from 'package:ggplot2':  
##   
## %+%  
##   
## Loading required package: qdapTools

## Warning: package 'qdapTools' was built under R version 3.2.4

##   
## Attaching package: 'qdapTools'  
##   
## The following object is masked from 'package:data.table':  
##   
## shift  
##   
##   
## Attaching package: 'qdap'  
##   
## The following object is masked from 'package:stringr':  
##   
## %>%  
##   
## The following object is masked from 'package:wordnet':  
##   
## synonyms  
##   
## The following objects are masked from 'package:tm':  
##   
## as.DocumentTermMatrix, as.TermDocumentMatrix  
##   
## The following object is masked from 'package:NLP':  
##   
## ngrams  
##   
## The following object is masked from 'package:base':  
##   
## Filter

* 1. Setting Up Wordnet for Use

As a second step, we needed to go online and download Wordnet to our local machine so that we could access it for the part of speech tagging (see section 3.). After Wordnet was downloaded successfully, the main dictionary folder was copied to the working directory of the R program so the following R code could be ran:

# For wordnet package setDict(file.path('C:','Program Files, (x86)', 'WordNet', #'2.1','dict'))  
setDict("dict")

This allows the certain functions in the NLP, Textresuse, and qdap packages to access the database.

* 1. Sourcing in the Functions and Loading the Data

The last step before we could even being going through the data was to load our functions file and load the data:

# Load in Functions  
source("project3Functions.R")

# Load Data  
data(acq)

1. Trying to Make Sense of the Original 50 Document acq Corpus
   1. Cleaning the Corpus

In order for us to be able to analyze the corpus, we first had to clean it up. This involved the following steps:

* Transforming everything to lowercase so that words that start a sentence are treated as the same word as one in a sentence
* Removing punctuation to get rid of possessive and to get similar parts of speech. This also makes sure that words that end a sentence are treated the same as words in the middle of a sentence.
* Removing numbers, which will get in the way of the analysis on text
* Removing English Stop words, which are commonly used words that will get in the way of the analyzing the documents (see below)
* Removing extra white space between words

English Stop Words from running **stopwords(kind = 'en')**:

[1] "i" "me" "my" "myself" "we"

[6] "our" "ours" "ourselves" "you" "your"

[11] "yours" "yourself" "yourselves" "he" "him"

[16] "his" "himself" "she" "her" "hers"

[21] "herself" "it" "its" "itself" "they"

[26] "them" "their" "theirs" "themselves" "what"

[31] "which" "who" "whom" "this" "that"

[36] "these" "those" "am" "is" "are"

[41] "was" "were" "be" "been" "being"

[46] "have" "has" "had" "having" "do"

[51] "does" "did" "doing" "would" "should"

[56] "could" "ought" "i'm" "you're" "he's"

[61] "she's" "it's" "we're" "they're" "i've"

[66] "you've" "we've" "they've" "i'd" "you'd"

[71] "he'd" "she'd" "we'd" "they'd" "i'll"

[76] "you'll" "he'll" "she'll" "we'll" "they'll"

[81] "isn't" "aren't" "wasn't" "weren't" "hasn't"

[86] "haven't" "hadn't" "doesn't" "don't" "didn't"

[91] "won't" "wouldn't" "shan't" "shouldn't" "can't"

[96] "cannot" "couldn't" "mustn't" "let's" "that's"

[101] "who's" "what's" "here's" "there's" "when's"

[106] "where's" "why's" "how's" "a" "an"

[111] "the" "and" "but" "if" "or"

[116] "because" "as" "until" "while" "of"

[121] "at" "by" "for" "with" "about"

[126] "against" "between" "into" "through" "during"

[131] "before" "after" "above" "below" "to"

[136] "from" "up" "down" "in" "out"

[141] "on" "off" "over" "under" "again"

[146] "further" "then" "once" "here" "there"

[151] "when" "where" "why" "how" "all"

[156] "any" "both" "each" "few" "more"

[161] "most" "other" "some" "such" "no"

[166] "nor" "not" "only" "own" "same"

[171] "so" "than" "too" "very"

The first step of converting the corpus to lower was done by itself in the code:

# Convert to Lower Case  
myCorp <- tm\_map(acq, content\_transformer(tolower))

While the remaining cleaning tasks were combined into one function called **cleanCorpus**:

cleanCorpus <- function(corp, extraStopWords){

## This function will do the following:

# 1: Remove Punctuation

# 2: Remove Numbers

# 3: Remove "stop" words and any extra words

# provided by the user

# 4: Remove extra whitespace

if (missing(extraStopWords)) extraStopWords <- ""

stopifnot(is.character(extraStopWords))

corpClean <- tm\_map(x = corp, FUN = PlainTextDocument) %>%

tm\_map(x = ., FUN = removePunctuation) %>%

tm\_map(x = ., FUN = removeNumbers) %>%

tm\_map(x = ., FUN = removeWords, c(stopwords(kind = 'en'),

extraStopWords)) %>%

tm\_map(x = ., FUN = stripWhitespace)

invisible(corpClean)

}

In the end, we originally tried to include the stemming function as well to make sure that the same word that only had different suffixes would be combined into one word. For instance, the word company and companies would be combined into the same word for analysis since in theory they are the same word. However, the stemming function was returning odd output because it seems that the first version that word is taken as the stem so the word company actually turned into “compani" (with the “i" at the end) because it stemmed companies to remove the “es”. Therefore, we thought it was best if we left the stemming function out of the cleaning corpus because we rather be able to manually change words in the corpus ourselves than have the stemming function do that for us.

* 1. Creating the Dendrogram, Wordcloud, and Term Frequency Bar Chart

Now that the data was cleaned, we were able to do the analysis on the corpus. This involved running the corpus through the functions that we learned in the Lecture on Text Mining as well as some other functions researched on our own. All of those functions were wrapped up into one function called **buildOutputs**, which ultimately created 4 outputs: a dendrogram, a wordcloud, a list of the most frequent terms, and a term frequency bar chart.

This function can be seen here:

buildOutputs <- function(corp,

minWordFreq = 10,

sparsity = .75,

docName = "Full ACQ Corpus"){

# Creates several output based on

# minimum freq and sparsity.

# This function will do the following:

# 1. Build dendrogram

# 2. Build wordcloud

# 3. Build bar chart of freq terms

# Find Frequent Terms and Sparse Matrix

myCorpTDM <- TermDocumentMatrix(corp,

control = list(wordLengths = c(1, Inf)))

wFreq <- sort(rowSums(as.matrix(myCorpTDM)), decreasing = T)

if(max(wFreq) < minWordFreq){

cat("Minimum word frequency filter (minWordFreq: ",

minWordFreq,

") is greater than the maximum word frequency.",

"\nSetting minWordFreq to",

"maximum word frequency:",

max(wFreq))

minWordFreq <- wFreq

}

wFreq <- subset(wFreq, wFreq >= minWordFreq)

if(is.null(wFreq)) stop("minWordFreq too large")

myCorpSparseTDM <- removeSparseTerms(myCorpTDM, sparsity)

myCorpDistMatrix <- dist(scale(myCorpSparseTDM))

# Dendrogram

DistMatrixHclust <- hclust(d = myCorpDistMatrix, method = 'ward.D2')

plot(DistMatrixHclust,

main = paste0('Cluster Dendrogram for ',

docName,

': Ward Scaled Distance'),

xlab = '', ylab = '', sub = '')

# Word Cloud

pal <- brewer.pal(6, "RdYlGn")

wordcloud(names(wFreq), wFreq, random.order = FALSE,

scale = c(2, .3), colors = pal)

# Find the Most Frequent Terms

fOfTerms <- findFreqTerms(myCorpTDM, minWordFreq)

cat("Terms Greater than ", minWordFreq, " Freq:\n\n")

print(fOfTerms)

# Bar Chart of Most Frequent Terms

myCorpDf <- data.table(term = names(wFreq),

freq = wFreq)

myCorpDf[, termSort := factor(term, levels = term[order(freq)])]

freqBar <- ggplot(myCorpDf, aes(x = termSort, y = freq)) +

geom\_bar(stat = "identity") +

xlab("Terms") +

ylab("Frequency") +

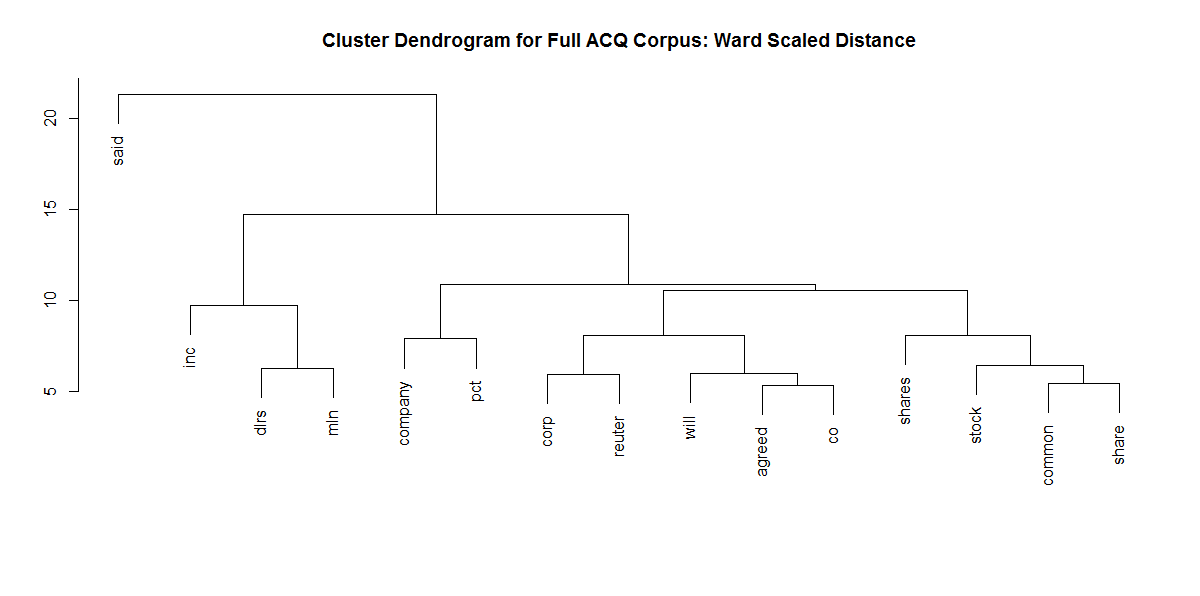
ggtitle(paste("Most Frequent Terms of", docName)) +

coord\_flip()

print(freqBar)

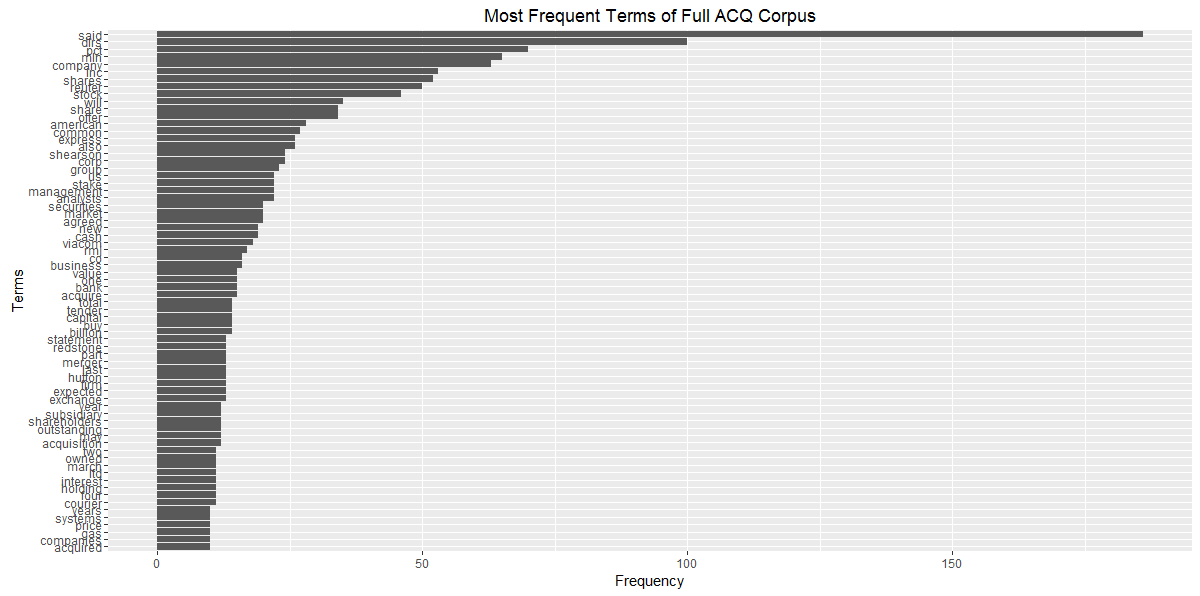
}

And for the entire corpus, the output was as follows (accounting for a sparsity of .75 when removing sparsity from the Term-Document matrix:



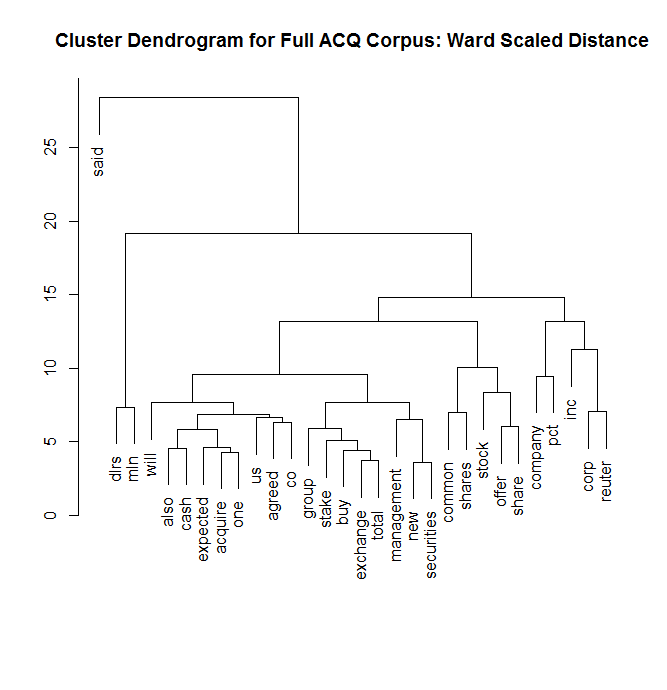


## Terms Greater than 10 Freq:  
##   
## [1] "acquire" "acquired" "acquisition" "agreed"   
## [5] "also" "american" "analysts" "bank"   
## [9] "billion" "business" "buy" "capital"   
## [13] "cash" "co" "common" "companies"   
## [17] "company" "corp" "courier" "dlrs"   
## [21] "exchange" "expected" "express" "firm"   
## [25] "four" "gas" "group" "holding"   
## [29] "hutton" "inc" "interest" "last"   
## [33] "ltd" "management" "march" "market"   
## [37] "may" "merger" "mln" "new"   
## [41] "offer" "one" "outstanding" "owned"   
## [45] "part" "pct" "price" "redstone"   
## [49] "reuter" "rmj" "said" "securities"   
## [53] "share" "shareholders" "shares" "shearson"   
## [57] "stake" "statement" "stock" "subsidiary"   
## [61] "systems" "tender" "total" "two"   
## [65] "us" "value" "viacom" "will"   
## [69] "year" "years"



There are several key takeaways from this set of output. First and foremost, the most commonly seen word throughout the entire corpus is the word “**said**”. While this is informative in that we can guess that throughout many of the articles people are being quoted, it does little to explain what the actual documents are about. But as already mentioned, it does identify that that the detail in the documents probably come from a firsthand source since it seems that much of the information (which we can assume are facts/opinions) are coming from comments of other people.

The second interesting thing about these documents is that the second most prolific word is “**dlrs**” which is an abbreviation of dollars. Additionally, the words “**pct**”, “**mln**”, “**shares**”, and “**company**” are the next most common words, meaning that we can make an educated guess that these documents are about public companies on the stock market. More importantly is that if we take a closer look at some of the smaller terms in the wordcloud, we can see other important terms such as “**aquire**”, “**buy**”, “**offer**”, and “**exchange**”. This means that not only are the documents about public companies that are traded on the stock market but it is specifically about buying of shares on the stock market and of potentially acquiring new companies. In fact if we increase the sparsity amount to .80 so that we keep more words for the dendrogram, we can see these terms show up on the graphic as well as their relationship between one another:



Another factor that we can see in these outputs is that the same stemmed word is showing up in several places. For instance, “company”, “corp”, “co”, “inc” and “firm” are all terms for a company (or corporation/incorporated) that are all the similar idea. Additionally, there are the words “share” and “shares”; “acquire”, “acquired”, and “acquisition”; and “year” and “years” in the output for the most frequent terms. With that in mind, it would behoove us to clean up the corpus a little more so that these words are combined into the same word due to being synonyms. Also, since there seem to be unnecessary words such as “will”, “said”, and “also”, those unnecessary words will be removed.

* 1. Re-cleaning the Corpus

1. Conclusion
   1. What we Learned about Data Science

Throughout this project we learned a lot about the idea of Exploratory Data Analysis (EDA) in regards to data science. First and foremost, there is not one definitive way to do any analysis and in fact as we saw during our analysis, there can be times when the results of one analysis can be ambiguous and inconclusive. In fact, we learned that to truly understand the results of the analysis and more specifically the underlying data we really need to run several different algorithms and compare results to make sure that each algorithm is saying consistent things before conclusions can truly be drawn.

Moreover, not only are there different algorithms that do relatively similar things but a good data scientist also understands how to use algorithms that are different and can complement each other. For instance, although we did not include the results in this write up we also tested the K-Nearest-Neighbor (KNN), K-modes and CART algorithms. The main reason we did not include the analyses from these algorithms was because KNN did not actually cluster the data but instead gave us results of how predictive our training data was in classifying edible or poisonous in our test data; K-modes is actually designed for categorical data and we thought it may bias our results to not use the same PCA data in all of our analyses, and CART did not allow us to specify the number of nodes on the tree, which we can deem as clusters and thus would be incomparable to the three methods we chose to write about here (K-means, Mclust, and PAM).

We also learned that data science may not be 100% correct, but can give the user a path to set on in continuing their analysis. For instance, by reducing the dimensionality of the data from 21 variables to 6 principal components, we essentially lost about 25% of the variance of the original dataset but gained a better understanding of what components were actually similar. This helped us identify which attributes were the most important such as Stalk Color (both above and below) and Population instead of Ring Type or Bruises. That way we could spend more of our time looking at fewer components and as we noticed early on with some algorithms, was essential for the algorithm to run in a convenient enough time so that we didn’t have to wait until the next day for the analysis to be ready (the k-modes algorithm was significantly faster running on the fewer variables, from overnight to 1.5 hours).

Overall, data science provides a multitude of tools designed to gain insights from data. Each tool has its advantages and disadvantages depending on the size and structure of the data as well as the goal of the analysis. It takes a great deal of exploration to determine the usefulness of each method, but it also requires common sense and analytical talent to extract information from data.

* 1. Final Thoughts on the Unknown Mushrooms

In the previous we mentioned that as we worked our way through this analysis, no one method 100% enabled us to identify the unknown mushrooms. It is our final thought that in order to identify the unknown mushrooms we would need to merge the results of each of our clustering methods. To do this we would take the results from the k-means and the PAM algorithm, specifically the poisonous mushrooms that fall into the edible dominant clusters from a 4 cluster split and compare them to the mushrooms that fall into the one cluster that from the Mclust 4 cluster split that has both edible and poisonous mushrooms. Using this process of comparing results between the different clustering algorithms we can validate which mushrooms labeled as poisonous should actually be unknown.

Additionally, as stated earlier in our analysis, we also identified that the most important variables for identifying unknown mushrooms from poisonous mushrooms are Stalk Color, both above and below, and population. These variables contributed to the majority of principal components 1 and 2, which explain the majority of the variance of the original data. Additionally, when looking at the pairwise plots between each attribute from the Mclust output, we can clearly see that these two attributes contain the most separated clusters meaning that the distance between data points are mostly in those dimensions and not in other dimensions.