Sentiment Analysis
Twitter Proof of Concept for Surveys: Avengers Endgame

Michael Vatt

10 Aug 20

Contents

Installation

```
# Create a Function to Check for Installed Packages and Install if They Are Not Installed

install <- function(packages){
  new.packages <- packages[!(packages %in% installed.packages()[, "Package"])]
  if (length(new.packages))
    install.packages(new.packages, dependencies = TRUE)
  sapply(packages, require, character.only = TRUE)
}

# Install
```

```
packages <- c("caret", "corpus", "data.table", "dendextend", "doParallel", "dplyr", "e1071",
              "ggplot2", "ggthemes","knitr", "magrittr", "petro.One", "plotly", "plotrix", "qdap",
              "randomForest", "RColorBrewer", "rlist", "RWeka", "scales", "SentimentAnalysis",
              "sentimentr", "SnowballC", "stringr", "syuzhet", "tidyr", "tidytext", "tidyverse",
              "tm", "viridisLite", "wordcloud", "xlsx")

install(packages)
```

```
# Call the installed packages

loadApp <- function() {

my_library <- c( "caret", "corpus", "data.table", "dendextend", "doParallel", "dplyr", "e1071",
                 "ggplot2", "ggthemes", "knitr", "magrittr", "petro.One",  "plotly", "plotrix",
                 "qdap", "randomForest", "RColorBrewer", "rlist", "RWeka", "scales",
                 "SentimentAnalysis", "sentimentr", "SnowballC", "stringr", "syuzhet", "tidyr",
                 "tidytext", "tidyverse", "tm", "viridisLite", "wordcloud", "xlsx")

install.lib <- my_library[!my_library %>% installed.packages()]

for(lib in install.lib) install.packages(lib, dependencies = TRUE)

sapply(my_library, require, character = TRUE)

}

loadApp()
```

Overview

This white paper was motivated by DIA's use of surveys as a major cost to the organization. Collecting feedback is critical to an organization's success, however, simply receiving feedback without translation and examination is uninformative and inconsequential. Therefore, surveys necessitate extensive labor hours to comb through their responses. This includes evaluation of possible classified information, the need to redact pieces of information, extreme language, and basic level exploratory analysis. The sentiments from open-ended questions require much more effort than those based upon Likert scales or Yes/No questions. Individuals are required to read each comment one by one. Resulting in the possible introduction of bias, inefficiencies, and inconsistencies to the area of sentiment analysis (SA). The goal is to provide a benefit to the Office of Human Resources (OHR) through this proof of concept.

Fortunately, the advancements of machine learning (ML) and artificial intelligence (AI) have developed and curated techniques and sometimes curated lexicons to evaluate sentiments of items comparable to surveys (e.g., Twitter tweets, Amazon Reviews). Combing through survey data and its respective comments is analogous to evaluating Twitter sentiments - the processes and methodologies are parallel and commensurate.

Introduction

This ML algorithm is a proof of concept utilizing *real* Twitter data, in the form of tweets, as an analogous concept to survey data. Both can be large data sets populated with open-ended text statements utilizing natural language and emotional expressions. This dataset was motivated by exploratory data analysis to see

if it may improve OHR processes in responding to the amount of the agency's survey data and its importance, in multiple structures and platforms compared to historical and laborious performance methods.

Advances in ML have permitted a fast and efficient way of combing natural language to detect emotions, keywords, ideas, and sentiment. While we will explore all of those, sentiment is the desired target information for this white paper.

What is Sentiment Analysis?

SA involves ML to computationally analyze writings (i.e., survey comments, tweets, opinions) and categorize their expressions as "positive", "negative", or "neutral". These models can aid any Human Resources office to determine how well the agency is performing - possibly identifying defects, recommended improvements, or strategy issues.

Part of this white paper is to determine if the ML model can accurately predict a comment's sentiment.

Classically, SA was done through active, but laborious, surveys - in-person and telephonic were the most common. As computers became more mainstream through advancements, so did SA - we just did not recognize it as such. This was done through simple surveys and simple calculations. Open-ended questions necessitated primitive methods to tackle the individual circumstances of every single survey collected. Open-ended questions lack the ability for true and consistent structure - due to the apparent stochastic behavior of respondents - processing of natural language demanded improved capabilities and techniques.

Model Design

This model's will draw from 15,000 tweets with reference to the box-office hit *Avengers: Endgame*.

Evaluating a collection of Tweets from Twitter may show a deeper, more accurate representation of sentiment regarding specific topics that would be difficult to collect elsewhere - if not impossible.

Lexicons have been developed and curated via open-source methods that aid in the discovery and investigation of SA on data sets - here, we will use *syuzhet*, *afinn*, *bing*, and *nrc*. Some may have less interoperability and are more restrictive.

From there, we will analyze the tweets, clean up the tweets in order to evaluate words that matter, remove unnecessary words, signs, numbers, and items like URLs.

WARNING:

This is an entirely *real* data set retrieved from Twitter. *Personally identifiable information (PII) may not be completely removed by the algorithm.* If PII remains in the data set, this is wholly unintentional. The data set is 100% publicly available information, thus, excluded from human subjects research protection requirements and free to use. In addition, PII is not the desired object of information and is therefore meaningless to this algorithm's outputs. However, be aware that this is real data from Twitter, and as such, there is always a risk that it may contain PII, profanity, or other offensive content.

This dataset includes the following variables:

variables1
Label
Text (Tweet)
senti (Sentiment)
s_score (Sentiment Score)

Objective

With the help of ML, our objective is to uncover possibilities of detecting sentiment from Twitter data (analogous to survey data). Ultimately resulting in meaningful input to the feedback system. There is a bias against using open-ended statements in feedback, possibly because they appear to be *noise* in the data or that they are *fruitless* and *inconvenient* data points. Due to their unstructured format and stochastic behavior, it creates a dilemma of weighing "pros-and-cons" to determine the benefit of spending the time to evaluate the statements. This white paper endeavors to discover the possibility of lowering labor costs, discovering hidden insights, and illustrate results that are both informative and substantive.

Dataset

```
split <- detectCores(TRUE)
split # To Make Sure it Worked
```

```
## [1] 24
```

```
cl <- makePSOCKcluster(split)
registerDoParallel(cl)
```

Let's download the data set from a .csv file and place it into a data frame:

```
tweet <- read.csv("https://raw.githubusercontent.com/mjvatt/Tweets/master/avengers%20tweets.csv",
                  stringsAsFactors = FALSE)
```

Methods and Analysis

Exploratory Data Analysis

Now let's gain some familiarization with the data set and view the first few rows.

```
# Check Initial Shape of Data Frame

dim(tweet)
```

```
## [1] 15000    17
```

```
# Remove Unnecessary Columns

tweet <- tweet[,-3:-17]

# What Shape is the Data Frame?

dim(tweet) # Dimensions of the Dataframe For Familiarity - This is Correct
```

```
## [1] 15000     2
```

```
## Structure of Data Frame

str(tweet) # Notice that Some Columns Are Not The Same Class
```

```
## 'data.frame':    15000 obs. of  2 variables:
##  $ X   : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ text: chr  "RT @mrvelstan: literally nobody:\nme:\n\n#AvengersEndgame https://t.co/LR9kFwfD5c" "R
```

```
class(tweet) # Let's Ensure it is Correct Format
```

```
## [1] "data.frame"
```

We now need to perform a little bit of pre-processing to change column names, character data types to factors, create a data partition, and start the sentiment analysis process.

```
# Column Names to Data Frame

colnames(tweet) <- c("label","text")

kable(tweet[4:8,], format = "markdown", align = "cc") # Does the Data Look Right? - Yes
```

	label	text
4	4	RT @HelloBoon: Man these #AvengersEndgame ads are everywhere https://t.co/Q0lNf5eJsX
5	5	RT @Marvel: We salute you, @ChrisEvans! #CaptainAmerica #AvengersEndgame https://t.co/VlPEpnXYgm
6	6	RT @MCU_Direct: The first NON-SPOILER #AvengersEndgame critic reactions are here and nearly all are exceptionally positive, with many prais...
7	7	RT @Renner4Real: Ready to rock ! #excited #avengersendgame #presstourcontinues #worldpremiere #endgame https://t.co/KXpKNJl9aq
8	8	RT @Avengers: We're with him 'til the end of the line. #WinterSoldier #AvengersEndgame https://t.co/Xi4cYqWgDR

```
# Changing Label From Char to Factor Form

tweet$label <- factor(tweet$label)

### Splitting Big Data File to 1/50 of i-th's Size

ind <- createDataPartition(tweet$label, p = 1/50, list = FALSE)
tweet1 <- tweet[ind,]
```

Sentiment Analysis

Let's evaluate those 5 example tweets from above and see if we can predict their individual sentiment qualities.

```
senti_analysis <- analyzeSentiment(tweet1$text)
kable(senti_analysis[1:5,-5:-11], digits = 3)
```

WordCount	SentimentGI	NegativityGI	PositivityGI	SentimentQDAP	NegativityQDAP	PositivityQDAP
5	0.000	0.000	0.000	0.000	0.000	0.000
6	-0.167	0.167	0.000	-0.167	0.167	0.000
6	0.167	0.000	0.167	0.167	0.000	0.167
5	0.200	0.000	0.200	0.000	0.000	0.000
6	0.333	0.000	0.333	0.333	0.000	0.333

```
sentiment <- convertToDirection(senti_analysis$SentimentQDAP)
tweet1$senti <- sentiment
```

Now let's use those sentiment scores to provide an overall sentiment per tweet.

```
tweet1$s_score <- senti_analysis$PositivityQDAP - senti_analysis$NegativityQDAP

kable(tweet1[4:8,], format = "markdown", digits = 2)
```

	label	text	senti	s_score
4	4	RT @HelloBoon: Man these #AvengersEndgame ads are everywhere https://t.co/Q0lNf5eJsX	neutral	0.00
5	5	RT @Marvel: We salute you, @ChrisEvans! #CaptainAmerica #AvengersEndgame https://t.co/VlPEpnXYgm	positive	0.33
6	6	RT @MCU_Direct: The first NON-SPOILER #AvengersEndgame critic reactions are here and nearly all are exceptionally positive, with many prais...	positive	0.09
7	7	RT @Renner4Real: Ready to rock ! #excited #avengersendgame #presstourcontinues #worldpremiere #endgame https://t.co/KXpKNJl9aq	positive	0.22
8	8	RT @Avengers: We're with him 'til the end of the line. #WinterSoldier #AvengersEndgame https://t.co/Xi4cYqWgDR	negative	-0.12

```
str(tweet1)
```

```
## 'data.frame':    15000 obs. of  4 variables:
##  $ label  : Factor w/ 15000 levels "1","2","3","4",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ text   : chr  "RT @mrvelstan: literally nobody:\nme:\n\n#AvengersEndgame https://t.co/LR9kFwfD5c"
##  $ senti  : Factor w/ 3 levels "negative","neutral",..: 2 1 3 2 3 3 3 1 3 2 ...
##  $ s_score: num  0 -0.167 0.167 0 0.333 ...
```

```
kable(round(prop.table(table(tweet1$senti)), 3))
```

Var1	Freq
negative	0.12
neutral	0.36
positive	0.52

Natural Language Processing

Here, we need to clean up the text that is in the tweets. This is more extensive than it may sound due to the esoteric nature of Twitter - involving specific syntax, high use of slang, or obtuse messaging.

This will use an example tweet to show the difference between an originally published tweet and a processed tweet.

First, in order properly utilize the words from the tweets, we have to create a Vector Source that will allow us to create a Volatile Corpora. This sounds worse than it is. It is simply how the data is stored on the computer. A Volatile Corpus (VCorpus) will use the RAM's memory rather than the disk memory (hard drive) of the computer as it would if we used a Permanent Corpus (PCorpus). Basically, it is a temporary object and the data will be erased when the object is destroyed - thus, is more efficient and less taxing.

A corpus is the plural of corpora, simply means a collection of written texts.

Now, we can clean the corpus of anything that is either unnecessary, not helpful, problematic, and/or not words. This includes items, such as: punctuation, URLs, unnecessary spaces, numbers, and we can lowercase everything to make the analysis much more efficient and simpler.

```
## Loading required package: NLP

corpus <- VCorpus(VectorSource(tweet1$text))

# A Snap Shot of the First Text Stored in the Corpus

comparison <- data.frame(1, stringsAsFactors = FALSE)

comparison[1,] <- as.character(corpus[[623]])

NumPunct <- function(x) gsub("[^[:alpha:][:space:]]*","",x)
removeURL <- function(x) gsub("http[^[:space:]]*","",x)
toSpace <- content_transformer(function (x , pattern ) gsub(pattern, " ", x))
corpus_clean <- tm_map(corpus, toSpace, "/")
corpus_clean <- tm_map(corpus_clean, toSpace, "@")
corpus_clean <- tm_map(corpus_clean, toSpace, "\\|")
corpus_clean <- tm_map(corpus_clean, content_transformer(tolower))
corpus_clean <- tm_map(corpus_clean, content_transformer(removeURL))
corpus_clean <- tm_map(corpus_clean, content_transformer(NumPunct))
corpus_clean <- tm_map(corpus_clean, removeNumbers)
corpus_clean <- tm_map(corpus_clean, removeWords, c(stopwords("en"), "tco", "avengersendgame",
                      "marvel", "avengers", "man", "premiere", "ads", "qlnfejsx", "helloboon",
                      "everywhere", "vlpepnxygm", "qnsmdcdm", "uufef", "going", "szfbsggq", "uff",
                      "httpstcoq", "ejsx", "the", "httpstco", "avengersendgam", "marvel",
                      "avengers", "man", "premiere", "ads", "ejsx", "helloboon", "lnf", "the",
                      "win", "originalfunko", "get", "paytm", "just", "httpstcos", "zfbsggq"))
corpus_clean <- tm_map(corpus_clean, removePunctuation)
corpus_clean <- tm_map(corpus_clean, stripWhitespace)
corpus_clean <- tm_map(corpus_clean, removeWords, letters)

# To Ensure Language is Still There and No Issues Have Arisen

comparison[2,] <- as.character(corpus_clean[[623]])
comparison[1,2] <- "Published Tweet: "
comparison[2,2] <- "Processed Tweet: "
```

```r
comparison <- comparison[, 2:1]
kable(comparison, format = "markdown")
```

V2	X1
Published Tweet:	RT @ErikDavis: Not going to spoil anything at all, but I will say I laughed, I cried, I cried, I cheered, I laughed, I cried again and then...
Processed Tweet:	rt erikdavis spoil anything will say laughed cried cried cheered laughed cried

We can see above the condition from the published tweet to the processed tweet has changed quite a bit. It may also seem strange and nonsensical when read normally - that is ok. We are removing words that either provide no help, meaningless, or are far too common (e.g., 'the'). Thus, the leftover words have some sort of sentiment, whether it is positive, negative, or neutral.

Below, we will build the term-document matrix (TDM) in order to identify the frequency of words in the data set. The TDM will also be utilized in order to find out how many terms and documents we have in the matrix and determine what the sparsity of those terms are. Once this is complete, we can remove sparse terms - meaning, the removal of words that have very little meaning or input into the SA.

A TDM is a way of representing the words in the text as a table (or matrix) of numbers. The rows of the matrix represent the text responses to be analyzed, and the columns of the matrix represent the words from the text that are to be used in the analysis. The most basic version is binary.

The function, *removeSparseTerms()*, simply refers to the threshold of *relative document frequency* for a term, *above which* the term will be removed.

```r
# Build a Term-Document Matrix

corpus_clean_tdm <- TermDocumentMatrix(corpus_clean)
tdm_sparse <- removeSparseTerms(corpus_clean_tdm, 0.99)
tdm_matrix <- as.matrix(tdm_sparse)

inspect(corpus_clean_tdm)
```

```
## <<TermDocumentMatrix (terms: 6956, documents: 15000)>>
## Non-/sparse entries: 98194/104241806
## Sparsity           : 100%
## Maximal term length: 34
## Weighting          : term frequency (tf)
## Sample             :
##                 Docs
## Terms            11205 11382 11644 11884 12185 1907 3251 4288 6083 8276
##   captainamerica     0     0     0     0     0    0    0    0    0    0
##   chance             0     0     0     0     0    0    0    0    0    0
##   chris              2     2     2     2     2    2    2    0    2    2
##   chrisevans         0     0     0     0     0    0    0    0    0    0
##   cried              0     0     0     0     0    0    0    0    0    0
##   johansson          1     1     1     1     1    1    1    0    1    1
##   like               0     0     0     0     0    0    0    0    0    0
##   salute             0     0     0     0     0    0    0    0    0    0
##   scarlett           1     1     1     1     1    1    1    0    1    1
##   world              0     0     0     0     0    0    0    0    0    0
```

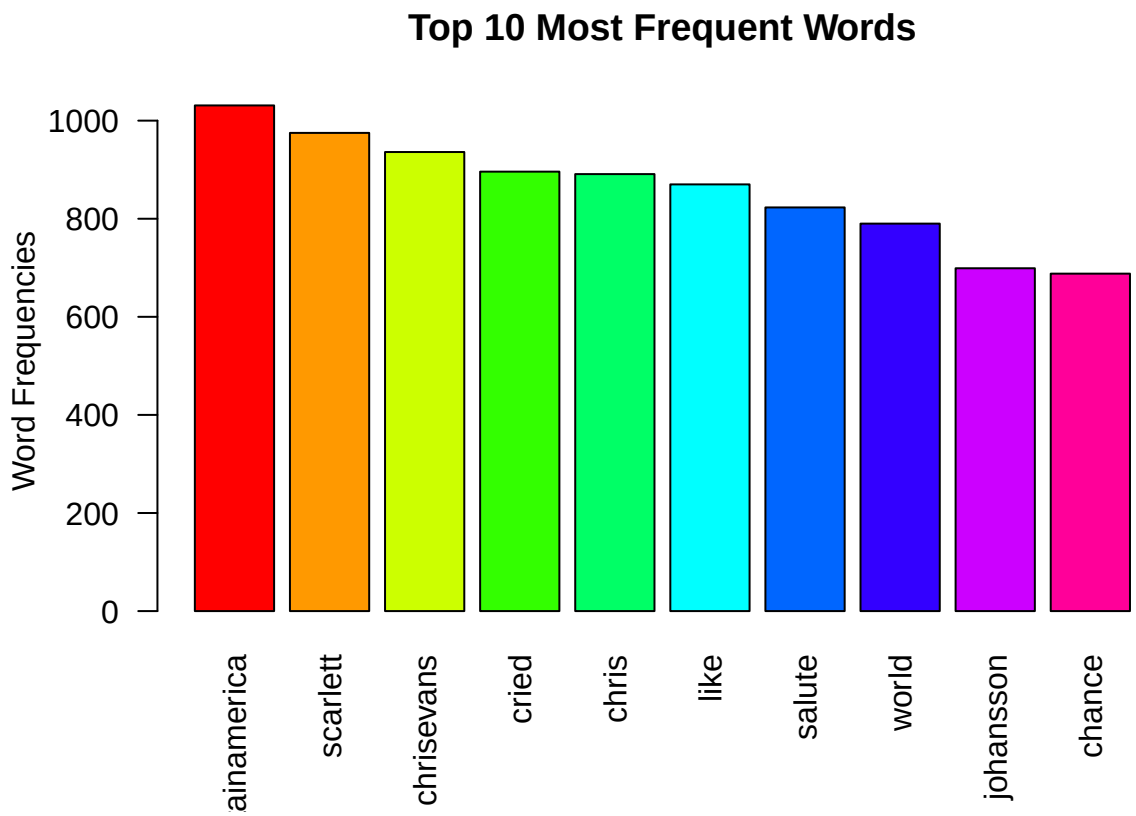
```
inspect(tdm_sparse)
```

```
## <<TermDocumentMatrix (terms: 144, documents: 15000)>>
## Non-/sparse entries: 46038/2113962
## Sparsity           : 98%
## Maximal term length: 16
## Weighting          : term frequency (tf)
## Sample             :
##                 Docs
## Terms            183 23 234 320 327 399 43 458 520 57
##   captainamerica   0  0   0   0   0   0  0   0   0  0
##   chance           0  0   0   0   0   0  0   0   0  0
##   chris            1  1   1   1   1   1  1   1   1  1
##   chrisevans       0  0   0   0   0   0  0   0   0  0
##   cried            2  2   2   2   2   2  2   2   2  2
##   johansson        0  0   0   0   0   0  0   0   0  0
##   like             0  0   0   0   0   0  0   0   0  0
##   salute           0  0   0   0   0   0  0   0   0  0
##   scarlett         0  0   0   0   0   0  0   0   0  0
##   world            0  0   0   0   0   0  0   0   0  0
```

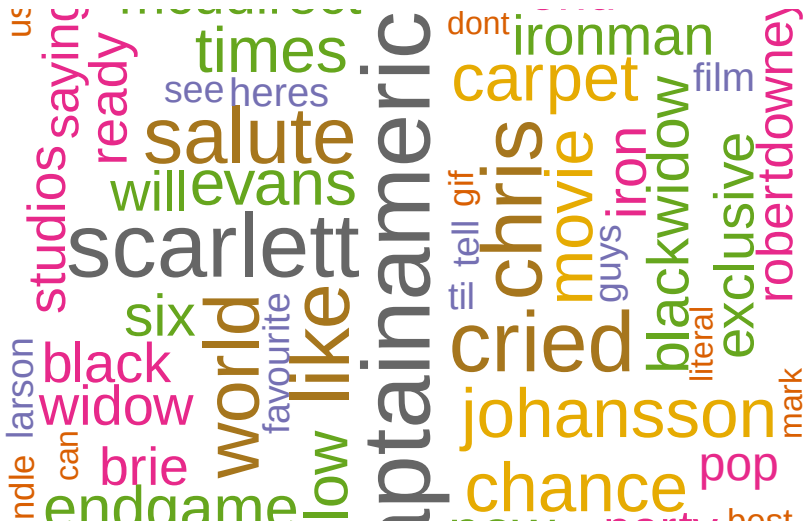
```
# Sort by Descending Value of Frequency

tdm_value <- sort(rowSums(tdm_matrix), decreasing = TRUE)
tdm_df <- data.frame(word = names(tdm_value), freq = tdm_value)

# Display Top 5 Most Frequent Words

kable(tdm_df[1:5,], format = "markdown")
```

	word	freq
captainamerica	captainamerica	1031
scarlett	scarlett	975
chrisevans	chrisevans	936
cried	cried	896
chris	chris	891

```
#head(tdm_df,5)

# Plot the Most Frequent Words

barplot(tdm_df[1:10,]$freq, las = 2, names.arg = tdm_df[1:10,]$word,
        col = rainbow(10), main ="Top 10 Most Frequent Words",
        ylab = "Word Frequencies")
```




```
# Generate Word Cloud

set.seed(1234)
wordcloud(words = tdm_df$word, freq = tdm_df$freq, min.freq = 5,
          max.words = 100, random.order = FALSE, rot.per = 0.40,
           scale = c(3,0.5), colors = brewer.pal(8, "Dark2"))
```



Next, we will use *findAssocs()* to test three arbitrary words to see whether there are words that are associated with them with a minimum correlation of 35% - if it were 100%, they would *always* be associated with each other.

After, we will find associations that occur at least 300 times with a minimum correlation of 50%.

```
# Find Associations of at least 35%

findAssocs(corpus_clean_tdm, terms = c("home","bed","stop"), corlimit = 0.35)
```

```
## $home
##     planet    gamoras        zen zenwhoberi
##       0.83       0.69       0.38       0.36
##
## $bed
## jumpedforjoi      offline       others
##         0.71         0.58         0.58
##
## $stop
##         leaves      lmnsgrdhs          wreck aayegatomodihi alreadymadehis
##           0.58           0.58           0.58           0.41           0.41
##          cares          causi      cchvzygqh      choicethe   dailyexpress
##           0.41           0.41           0.41           0.41           0.41
##      dcxnhxveu       glyzpblu        goddamn     kardashian            kim
##           0.41           0.41           0.41           0.41           0.41
```

```
##        lveodjm     mejorityno            sin  sophiabelrose          swamy
##           0.41           0.41           0.41           0.41           0.41
##          voter
##           0.41
```

```
# Find Associations For Words That Occur at Least 300 Times and at least 50%

findAssocs(corpus_clean_tdm, terms = findFreqTerms(corpus_clean_tdm, lowfreq = 300)[8:15],
           corlimit = 0.50)
```

```
## $captainmarvel
## adytpridi
##       0.7
##
## $carpet
##  red
## 0.72
##
## $chance
##        follow boxlunchgifts funkoavengers           pop     exclusive
##          0.85          0.76          0.76          0.76          0.67
##          iron
##          0.67
##
## $chris
##     evans hemsworth       six     times
##      0.82      0.63      0.51      0.50
##
## $chrisevans
##         salute captainamerica
##           0.93           0.87
##
## $cried
##     times       six   literal    saying  targrycn  honestly     means hemsworth
##      0.80      0.75      0.74      0.73      0.73      0.72      0.67      0.55
##   laughed     evans
##      0.52      0.50
##
## $end
##          line           til     xicyqwgdr wintersoldier       oofnxkz
##          0.75          0.75          0.56          0.53          0.51
##
## $endgame
## numeric(0)
```

Now, we will employ the *syuzhet* package. This has abilities to extract sentiment and sentiment-derived plot arcs from text using a variety of sentiment dictionaries conveniently packaged. Syuzhet was developed by the Nebraska Literary Lab and is the default for extracting sentiment in the package. The three others are : *afinn* developed by Finn Nielsen, *bing* developed by Minqing Hu and Bing Liu, and *nrc* developed by Saif Mohammad and Peter Turney of the National Research Council of Canada. The package provides several methods for plot arc normalization.

The *syuzhet* package attempts to reveal the latent structure of narratives by means of SA. Instead of detecting shifts in the topic or subject matter of the narrative, it reveals the emotional shifts that serve as

proxies for the narrative movement.

```
# Regular Sentiment Score Using get_sentiment() Function and Others
# NOTE: Different Methods May Have Different Scales

tweet2 <- as.character(tweet)
tweet_sub2 <- tweet2[2]
syuzhet_vector <- get_sentiment(tweet2, method = "syuzhet")

# See the First Row of the Vector

syuzhet_vector
```

```
## [1]  0.00 68.75
```

```
# See Summary Statistics of the Vector

summary(syuzhet_vector)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00   17.19   34.38   34.38   51.56   68.75
```

The *get_sentiment()* function will assess the sentiment of each word or sentence. It takes two arguments: a character vector and a method of evaluation. The method will determine which of the sentiment lexicon methods to employ. Unfortunately the Stanford method was sidelined due to persistent errors and not used for this concept.

Each method will evaluate the sentiment with a slightly different scale and may return different results. Therefore, we should compare the results of their overall sign to check for large discrepancies - if any exist.

```
# Bing

bing_vector <- get_sentiment(tweet2, method = "bing")
head(bing_vector)
```

```
## [1] 0 9
```

```
summary(bing_vector)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00    2.25    4.50    4.50    6.75    9.00
```

```
# Affin

afinn_vector <- get_sentiment(tweet2, method = "afinn")
head(afinn_vector)
```

```
## [1]   0 134
```

```r
summary(afinn_vector)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.0    33.5    67.0    67.0   100.5   134.0
```

```r
# NRC

nrc_vector <- get_sentiment(tweet2, method = "nrc", lang = "english")
head(nrc_vector)
```

```
## [1]   0 101
```

```r
summary(nrc_vector)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00   25.25   50.50   50.50   75.75  101.00
```

Above, we can see that the syuzhet method returned (68.75), suggesting that the overall sentiment was really positive. The mean (34.375) and median (34.375) support this.

Bing returned much smaller numbers, max (9) mean (4.5) and median (4.5) are still positive.

Afinn returned larger numbers, max (134) mean (67) and median (67) are also positive.

NRC returned similar numbers, max (101) mean (50.5) and median (50.5) are also positive.

The signs of each succinctly show this as well below.

The fact that the min of each method were all zero show the distribution was overwhelmingly positive - granted these are tweets about the mega-hit Avengers: Endgame movie. Therefore, it could have been surmised that this was likely to be positive due to the topic being covered.

```r
# Compare the First Row of Each Vector Using Sign Function

rbind(
  sign(head(syuzhet_vector)),
  sign(head(bing_vector)),
  sign(head(afinn_vector)),
  sign(head(nrc_vector))
)
```

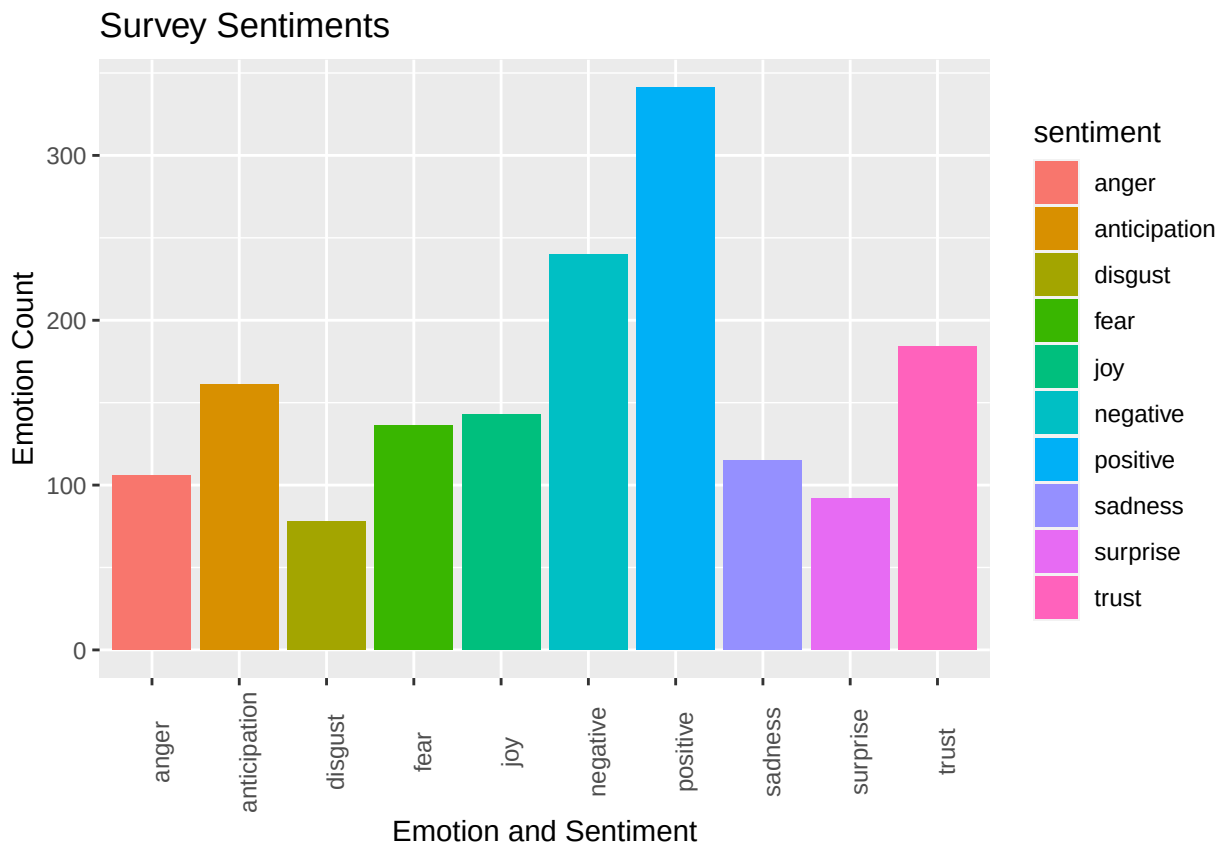
```
##      [,1] [,2]
## [1,]    0    1
## [2,]    0    1
## [3,]    0    1
## [4,]    0    1
```

But we can further evaluate the overall sentiment over each tweet individually (positive or negative) and classify each tweet with an overall emotion ranging from: Anger, Anticipation, Disgust, Fear, Joy, Sadness, Surprise, Trust.

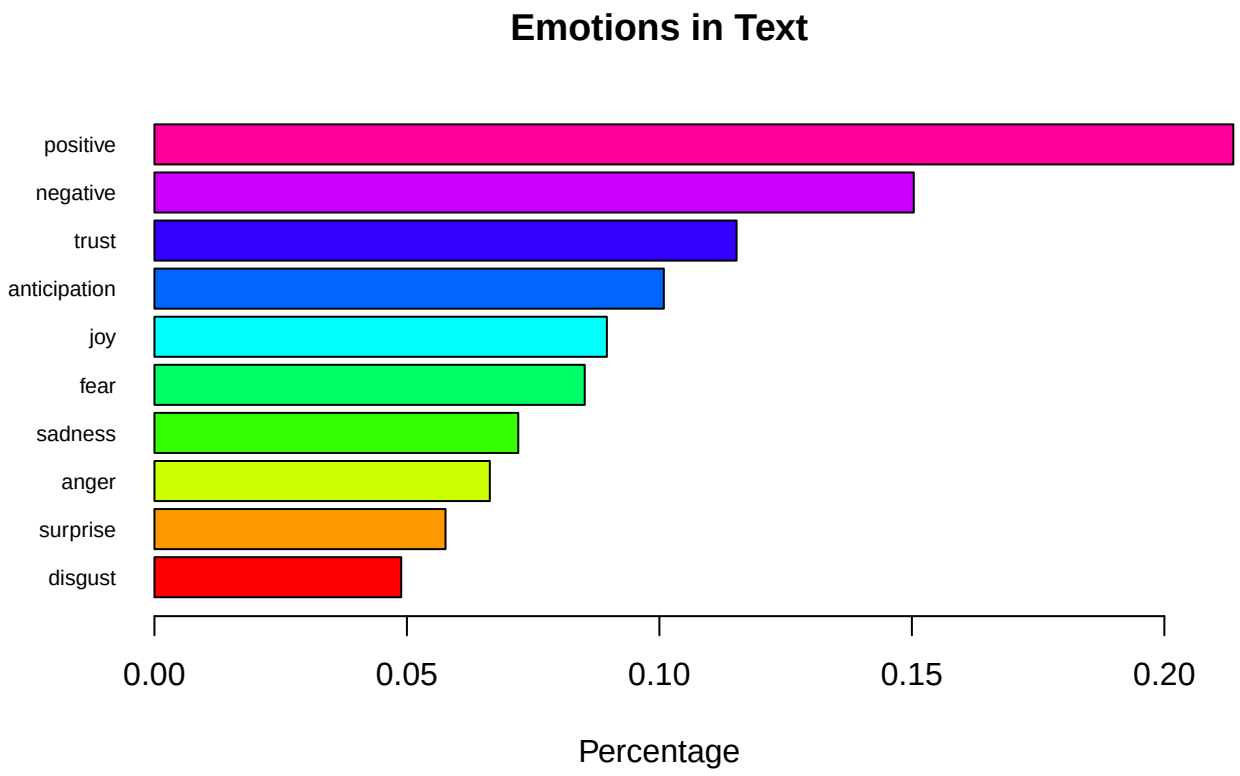
```r
# Run NRC Sentiment Analysis to Return Data Frame With Each Row Classified as One of the Following
# Emotions, Rather than a Score:
# Anger, Anticipation, Disgust, Fear, Joy, Sadness, Surprise, Trust
# It Also Counts the Number of Positive and Negative Emotions Found in Each Row

d <- get_nrc_sentiment(tweet_sub2)

# Head(d,10) - To See Top 10 Lines of the get_nrc_sentiment Data Frame

kable(head(d,10))
```

anger	anticipation	disgust	fear	joy	sadness	surprise	trust	negative	positive
106	161	78	136	143	115	92	184	240	341

```r
# Transpose

td <- data.frame(t(d))

# The Function rowSums Computes Column Sums Across Rows for Each Level of a Grouping Variable.

td_new <- data.frame(rowSums(td))

# Transformation and Cleaning

names(td_new)[1] <- "count"
td_new <- cbind("sentiment" = rownames(td_new), td_new)
rownames(td_new) <- NULL
td_new2<-td_new[1:10,]

# Plot One - Count of Words Associated With Each Sentiment

qplot(sentiment, data = td_new2, weight = count, geom = "bar", fill = sentiment,
      xlab = "Emotion and Sentiment", ylab = "Emotion Count") + ggtitle("Survey Sentiments") +
      theme(axis.text.x = element_text(angle = 90))
```




```
# Plot Two - Count of Words Associated With Each Sentiment, Expressed as a Percentage

barplot(sort(colSums(prop.table(d[, 1:10]))), horiz = TRUE, cex.names = 0.7, las = 1,
  main = "Emotions in Text", xlab = "Percentage", col = rainbow(10),
  xlim = c(0,.2))
```



Below we will see most frequent terms that appear more than 300 times. Followed by a breakdown of positive, negative, and neutral tweets.

```
# Different Predictions

dtm <- DocumentTermMatrix(corpus_clean)
dim(dtm)
```

```
## [1] 15000  6956
```

```
dtm <- removeSparseTerms(dtm, 0.99)
dim(dtm)
```

```
## [1] 15000   144
```

```
# Inspecting the the First 10 Tweets and 10 Words in the Dataset

freq <- sort(colSums(as.matrix(dtm)), decreasing = TRUE)

findFreqTerms(dtm, lowfreq = 300) # Identifying Terms that Appears More than 300 Times
```

```
##  [1] "amp"           "avenger"       "black"          "blackwidow"
##  [5] "boxlunchgifts" "brie"          "captainamerica" "captainmarvel"
##  [9] "carpet"        "chance"        "chris"          "chrisevans"
## [13] "cried"         "end"           "endgame"        "evans"
```

```
## [17] "exclusive"      "film"           "first"          "follow"
## [21] "funkoavengers"  "hemsworth"      "iron"           "ironman"
## [25] "johansson"      "like"           "marvelstudios"  "mcudirect"
## [29] "movie"          "now"            "one"            "party"
## [33] "pop"            "reactions"      "ready"          "red"
## [37] "robertdowneyjr" "salute"         "saying"         "scarlett"
## [41] "six"            "studios"        "thanos"         "thor"
## [45] "ticketnew"      "times"          "unyjgww"        "vlttllcqe"
## [49] "welcome"        "widow"          "will"           "world"
```

```
wf <- data.frame(word = names(freq), freq = freq)

# Word Clouds

tweet3 <- tweet1 %>% select(text, senti)

positive <- subset(tweet3, senti == "positive")
positive_count <- positive %>% summarise(n())
kable(positive[2:5,], format = "markdown")
```

	text	senti
5	RT @Marvel: We salute you, @ChrisEvans! #CaptainAmerica #AvengersEndgame https://t.co/VlPEpnXYgm	positive
6	RT @MCU_Direct: The first NON-SPOILER #AvengersEndgame critic reactions are here and nearly all are exceptionally positive, with many prais...	positive
7	RT @Renner4Real: Ready to rock ! #excited #avengersendgame #presstourcontinues #worldpremiere #endgame https://t.co/KXpKNJl9aq	positive
9	RT @Variety: #AvengersEndgame first reactions: 'Most emotional, most epic MCU film' https://t.co/w4cojZzhPl	positive

```
colnames(positive_count) <- c("Total Positive")
positive_count
```

```
##   Total Positive
## 1           7800
```

```
#wordcloud(positive$text, max.words = 100, scale = c(3,0.5), colors = brewer.pal(8, "Dark2"))

negative <- subset(tweet3, senti == "negative")
negative_count <- negative %>% summarise(n())
kable(negative[2:5,], format = "markdown")
```

	text	senti
8	RT @Avengers: We're with him 'til the end of the line. #WinterSoldier #AvengersEndgame https://t.co/Xi4cYqWgDR	negative
45	RT @BrandonDavisBD: While watching #AvengersEndgame, I laughed so loud, I clapped with so much force, and I cried so hard. The movie exceed...	negative
99	RT @Avengers: .@AnthonyMackie soars onto the carpet for some selfies! <U+0001F933> #Falcon #AvengersEndgame https://t.co/ul2FuLezsc	negative

	text	senti
116	RT @Avengers: It's #StarLord, man! @prattprattpratt #AvengersEndgame https://t.co/ZHod1J2W31	negative

```
colnames(negative_count) <- c("Total Negative")
negative_count
```

```
##   Total Negative
## 1           1797
```

```
#wordcloud(negative$text, max.words = 100, scale = c(3,0.5), colors = brewer.pal(8, "Dark2"))

neutral <- subset(tweet3, senti == "neutral")
neutral_count <- neutral %>% summarise(n())
kable(neutral[2:5,], format = "markdown")
```

	text	senti
4	RT @HelloBoon: Man these #AvengersEndgame ads are everywhere https://t.co/Q0lNf5eJsX	neutral
10	RT @HelloBoon: Man these #AvengersEndgame ads are everywhere https://t.co/Q0lNf5eJsX	neutral
11	RT @Avengers: Destiny has arrived, Josh Brolin! #Thanos #AvengersEndgame https://t.co/klb2Zrk0pr	neutral
17	RT @Avengers: We salute you, @ChrisEvans! #CaptainAmerica #AvengersEndgame https://t.co/S4ZfBSGGQ0	neutral

```
colnames(neutral_count) <- c("Total Neutral")
neutral_count
```

```
##   Total Neutral
## 1          5403
```

```
#wordcloud(neutral$text, max.words = 100, scale = c(3,0.5), colors = brewer.pal(8, "Dark2"))

set.seed(1234)
wordcloud(words = wf$word, freq = wf$freq, min.freq = 1,
          max.words = 200, random.order = FALSE, rot.per = 0.35,
          colors = brewer.pal(8, "Dark2"),
          scale=c(3,0.5))
```

Now, we can look towards predicting. Naive Bayes (NB) trains on categorical data. We will convert the numeric data by creating a function to convert any non-zero positive value to *yes* and all zero values to *no* to state whether a specific term is present in the document.

The text corpus is stored as a dataframe and will need to merge that with the sentiment variable.

From there we can split the data set into training and test sets.

```
convert_count <- function(x) {
  y <- ifelse(x > 0, 1,0)
  y <- factor(y, levels = c(0,1), labels = c("No", "Yes"))
  y
}

# Apply the Convert_Count Function to Get Final Training and Testing DTMs

datasetNB <- apply(dtm, 2, convert_count)

dataset <- as.data.frame(as.matrix(datasetNB))

dataset$Class <- tweet3$senti

set.seed(222)

# split <- createDataPartition(dataset$Class, times = 1, p = 0.3, list = FALSE)
# train_set <- dataset[-split,]
```

```r
# test_set <- dataset[split,]

split <- sample(2, nrow(dataset), prob = c(0.75,0.25), replace = TRUE)
train_set <- dataset[split == 1,]
test_set <- dataset[split == 2,]

kable(prop.table(table(train_set$Class)))
```

Var1	Freq
negative	0.1215690
neutral	0.3624934
positive	0.5159377

```r
kable(prop.table(table(test_set$Class)))
```

Var1	Freq
negative	0.1144091
neutral	0.3532110
positive	0.5323799

Random Forest is one of the well known and validated models available. It performed extremely well on this data set.

```r
rf_classifier <- randomForest(x = train_set[-1210], y = train_set$Class, ntree = 1000)

rf_classifier
```

```
##
## Call:
##  randomForest(x = train_set[-1210], y = train_set$Class, ntree = 1000)
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 12
##
##         OOB estimate of  error rate: 0.01%
## Confusion matrix:
##          negative neutral positive  class.error
## negative     1373       0        0 0.0000000000
## neutral         1    4093        0 0.0002442599
## positive        0       0     5827 0.0000000000
```

```r
# Predicting the Test Set Results

rf_pred <- predict(rf_classifier, newdata = test_set[-1210])

# Making the Confusion Matrix

confusionMatrix(table(rf_pred, test_set$Class))
```

```
## Confusion Matrix and Statistics
##
##
## rf_pred    negative neutral positive
##   negative      424       1        0
##   neutral         0    1308        0
##   positive        0       0     1973
##
## Overall Statistics
##
##                Accuracy : 0.9997
##                  95% CI : (0.9985, 1)
##     No Information Rate : 0.5324
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9995
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: negative Class: neutral Class: positive
## Sensitivity                   1.0000         0.9992          1.0000
## Specificity                   0.9997         1.0000          1.0000
## Pos Pred Value                0.9976         1.0000          1.0000
## Neg Pred Value                1.0000         0.9996          1.0000
## Prevalence                    0.1144         0.3532          0.5324
## Detection Rate                0.1144         0.3529          0.5324
## Detection Prevalence          0.1147         0.3529          0.5324
## Balanced Accuracy             0.9998         0.9996          1.0000
```

NB Classifier also performed very well. NB works on the assumption that the features of the dataset are independent of each other. Thus, giving it the name of "naive". It tends to work well for *bag-of-words* models like text documents because words are largely independent of each other. The location of a word does not typically depend on another word. Therefore, it is commonly used for text classifications, SA, spam filtering, and recommendation systems.

```
# Naive Bayes

control <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
classifier_nb <- naiveBayes(train_set, train_set$Class, laplace = 1,
                                          trControl = control, tuneLength = 7)

nb_pred <- predict(classifier_nb, type = 'class', newdata = test_set)

confusionMatrix(nb_pred, test_set$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction negative neutral positive
##   negative      418       1        0
```

```
##   neutral         5    1308        2
##   positive        1       0     1971
##
## Overall Statistics
##
##                Accuracy : 0.9976
##                  95% CI : (0.9954, 0.9989)
##     No Information Rate : 0.5324
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9958
##
##  Mcnemar's Test P-Value : 0.129
##
## Statistics by Class:
##
##                      Class: negative Class: neutral Class: positive
## Sensitivity                   0.9858         0.9992          0.9990
## Specificity                   0.9997         0.9971          0.9994
## Pos Pred Value                0.9976         0.9947          0.9995
## Neg Pred Value                0.9982         0.9996          0.9988
## Prevalence                    0.1144         0.3532          0.5324
## Detection Rate                0.1128         0.3529          0.5318
## Detection Prevalence          0.1131         0.3548          0.5321
## Balanced Accuracy             0.9928         0.9982          0.9992
```

```
svm_classifier <- svm(Class~., data = train_set)
svm_classifier
```

```
##
## Call:
## svm(formula = Class ~ ., data = train_set)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  7852
```

```
svm_pred <- predict(svm_classifier, test_set)

confusionMatrix(svm_pred, test_set$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction negative neutral positive
##   negative      190       4       15
##   neutral       189    1221      583
##   positive       45      84     1375
##
```

```
## Overall Statistics
##
##                Accuracy : 0.7518
##                  95% CI : (0.7375, 0.7656)
##     No Information Rate : 0.5324
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5775
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: negative Class: neutral Class: positive
## Sensitivity                  0.44811         0.9328          0.6969
## Specificity                  0.99421         0.6779          0.9256
## Pos Pred Value               0.90909         0.6126          0.9142
## Neg Pred Value               0.93309         0.9486          0.7284
## Prevalence                   0.11441         0.3532          0.5324
## Detection Rate               0.05127         0.3295          0.3710
## Detection Prevalence         0.05640         0.5378          0.4058
## Balanced Accuracy            0.72116         0.8054          0.8112
```

```
guess <- positive_count[1,]/(positive_count[1,] + negative_count[1,] + neutral_count[1,])
guess <- percent(guess)
```

The Support Vector Machine (SVM) algorithm performed the worst. SVM is a powerful algorithm that finds the hyperplane that differentiates the two classes to be predicted, nicknamed *ham* and *spam*, and it does it very well. SVM can also perform linear and non-linear classification problems. However, it performed poorly. It still performed better (44% improvement) than if we had simply guessed *positive* for every single tweet 52%.

Conclusion

Here, we ignored data like *usernames*, *time stamps*, if a tweet was *favorited*, *replied to*, *re-tweeted*, and details like *gps location*. These were ignored because they were not analogous or corresponding to data commonly found in survey data.

The prediction algorithms performed better than expected, but that may be due to this particular data set as they did not perform as well on another data set that was originally being analyzed. However, that data set proved to be problematic in other ways, thus was discarded as not helpful.

This proof of concept is just that, a proof of concept. It is not holistically complete and capabilities are not limited to what is within this paper. However, survey data will necessitate the need to create specific functions, modifications, and resourcefulness that the performance of, is not available here.

Overall, this is a useful example of where we can see how much ML and data science can be used in HR applications. At the very least, it can be run against survey feedback to simply see what emotions are commonly found and what extremes may exist that may have not been recognized without these sorts of algorithms. Hopefully, more can be done to evaluate the survey data, but at a minimum, this may highlight data points that might have remained obscured and concealed by the noise and layers of digging into open-ended questions.

```
sessionInfo()
```

```
## R version 3.6.3 (2020-02-29)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19041)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] xlsx_0.6.3              wordcloud_2.6           viridisLite_0.3.0
##  [4] tm_0.7-7                NLP_0.2-0               forcats_0.5.0
##  [7] purrr_0.3.4             readr_1.3.1             tibble_3.0.1
## [10] tidyverse_1.3.0         tidytext_0.2.5          tidyr_1.1.0
## [13] syuzhet_1.0.4           stringr_1.4.0           SnowballC_0.7.0
## [16] sentimentr_2.7.1        SentimentAnalysis_1.3-3 scales_1.1.1
## [19] RWeka_0.4-42            rlist_0.4.6.1           randomForest_4.6-14
## [22] qdap_2.4.1              RColorBrewer_1.1-2      qdapTools_1.3.5
## [25] qdapRegex_0.7.2         qdapDictionaries_1.0.7  plotrix_3.7-8
## [28] plotly_4.9.2.1          petro.One_0.2.3         magrittr_1.5
## [31] knitr_1.28              ggthemes_4.2.0          e1071_1.7-3
## [34] dplyr_1.0.0             doParallel_1.0.15       iterators_1.0.12
## [37] foreach_1.5.0           dendextend_1.13.4       data.table_1.12.8
## [40] corpus_0.10.1           caret_6.0-86            ggplot2_3.3.2
## [43] lattice_0.20-41
##
## loaded via a namespace (and not attached):
##  [1] readxl_1.3.1         backports_1.1.7      plyr_1.8.6
##  [4] igraph_1.2.5         lazyeval_0.2.2       splines_3.6.3
##  [7] openNLP_0.2-7        digest_0.6.25        htmltools_0.5.0
## [10] viridis_0.5.1        gender_0.5.4         fansi_0.4.1
## [13] gdata_2.18.0         recipes_0.1.12       modelr_0.1.8
## [16] gower_0.2.1          colorspace_1.4-1     blob_1.2.1
## [19] rvest_0.3.5          haven_2.3.1          xfun_0.15
## [22] crayon_1.3.4         RCurl_1.98-1.2       jsonlite_1.6.1
## [25] survival_3.2-3       glue_1.4.1           gtable_0.3.0
## [28] ipred_0.9-9          DBI_1.1.0            Rcpp_1.0.4.6
## [31] textclean_0.9.3      stats4_3.6.3         lava_1.6.7
## [34] prodlim_2019.11.13   htmlwidgets_1.5.1    httr_1.4.1
## [37] ellipsis_0.3.1       farver_2.0.3         pkgconfig_2.0.3
## [40] XML_3.99-0.3         rJava_0.9-12         openNLPdata_1.5.3-4
## [43] nnet_7.3-14          dbplyr_1.4.4         venneuler_1.1-0
## [46] utf8_1.1.4           labeling_0.3         tidyselect_1.1.0
```

```
## [49] rlang_0.4.6          reshape2_1.4.4       munsell_0.5.0
## [52] cellranger_1.1.0     tools_3.6.3          cli_2.0.2
## [55] generics_0.0.2       broom_0.5.6          evaluate_0.14
## [58] yaml_2.2.1           ModelMetrics_1.2.2.2 fs_1.4.1
## [61] nlme_3.1-148         slam_0.1-47          xml2_1.3.2
## [64] tokenizers_0.2.1     compiler_3.6.3       rstudioapi_0.11
## [67] reprex_0.3.0         stringi_1.4.6        highr_0.8
## [70] Matrix_1.2-18        RWekajars_3.9.3-2    vctrs_0.3.1
## [73] pillar_1.4.4         lifecycle_0.2.0      bitops_1.0-6
## [76] R6_2.4.1             gridExtra_2.3        janeaustenr_0.1.5
## [79] lexicon_1.2.1        codetools_0.2-16     MASS_7.3-51.6
## [82] gtools_3.8.2         assertthat_0.2.1     chron_2.3-55
## [85] xlsxjars_0.6.1       withr_2.2.0          hms_0.5.3
## [88] grid_3.6.3           rpart_4.1-15         timeDate_3043.102
## [91] class_7.3-17         rmarkdown_2.3        pROC_1.16.2
## [94] lubridate_1.7.9

## Time difference of 1.61821 mins
```