Text Sentiment Analysis Project

Praveen Srivatsa

January 9, 2020

SENTIMENT ANALYSIS
Discovering people opinions, emotions and feelings about a product or service

Figure 1: Sentiment Analysis

Introduction and Executive Summary

For any company, understanding the pulse of their customer is key. With the explosion of social media, people are posting their comments about all things in various places on the internet. Managing and keeping track of what people are talking about the company or its product online is an important part of today's corporate world. However, this is easier said than done.

Instead of trying to read all the posts, comments, tweets etc, companies can choose to use machine learning to understand the sentiment of the user posts and comments. Many companies are successfully leveraging sentiment analysis to understand user behaviours and preferences. Machine Learning techniques can be used to understand the sentiment against a product, a service or a company from the posts and comments about it online. Comments about a product on an online shopping service like Amazon, eBay or BigCommerce is a good indicator of the sentiment of users against specific products. Getting the sentiments of users from their comments against a song on iTunes or a movie on NetFlix is a great indicator of how users are reacting to these releases.

In this project, we take a look at how we can understand sentiment from text data that is collected from twitter. We could expand the same example to get the sentiment for movie reviews, airline sentiment or even just plain email sentiments. This can give a very good birds eye view of understanding sentiment from a large pool of digital comments across the globe.

Methods and techniques

Let us first understand the methods and techniques of working with sentiment analysis with text data. There are two broad ways of working with text mining and sentiment analysis.

Lets set up the packages for our project. (Note : if you dont already have these packages installed, you might have to run this twice to install and load it)

```
if(!require(tidyverse))
  install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(tidytext))
  install.packages("tidytext", repos = "http://cran.us.r-project.org")
if(!require(textdata))
  install.packages("textdata", repos = "http://cran.us.r-project.org")
if(!require(tm))
  install.packages("tm", repos = "http://cran.us.r-project.org")
if(!require(SnowballC))
  install.packages("SnowballC", repos = "http://cran.us.r-project.org")
if(!require(ggplot2))
  install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(wordcloud))
  install.packages("wordcloud", repos = "http://cran.us.r-project.org")
if(!require(caret))
  install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(randomForest))
  install.packages("randomForest", repos = "http://cran.us.r-project.org")
```

Simple Text mining and Sentiment Analysis

Basic text mining has been discussed in many forumns and blogs. In this, the text is broken up into independent words and this is then mapped againt a *dictionary* of sentiment words. Lets take a quick look at how this works.

```
poem <- c("Roses are red,", "Violets are blue,","Sugar is sweet,", "And so are you.")
example <- tibble(line = c(1, 2, 3, 4),text = poem)
example
```

```
## # A tibble: 4 x 2
##    line text
##   <dbl> <chr>
## 1     1 Roses are red,
## 2     2 Violets are blue,
## 3     3 Sugar is sweet,
## 4     4 And so are you.
```

We take this simple poem and tokennize it

```
example_words <- example %>% unnest_tokens(word, text)
example_words %>%
  count(word) %>%
  arrange(desc(n))
```

```
## # A tibble: 11 x 2
##    word        n
##    <chr>   <int>
##  1 are         3
##  2 and         1
##  3 blue        1
##  4 is          1
##  5 red         1
##  6 roses       1
##  7 so          1
##  8 sugar       1
##  9 sweet       1
## 10 violets     1
## 11 you         1
```

Then we download and open a sentiments dictionary. There are multiples ones like *bing*, *afinn*, *nrc* or *loughran*. Here lets use the nrc library and join this with our list of words to figure out the sentiment for the words in the poem.

```
nrc <- get_sentiments("nrc") %>%
  select(word, sentiment)
nrc
```

```
## # A tibble: 13,901 x 2
##    word        sentiment
##    <chr>       <chr>
##  1 abacus      trust
##  2 abandon     fear
##  3 abandon     negative
##  4 abandon     sadness
##  5 abandoned   anger
##  6 abandoned   fear
##  7 abandoned   negative
##  8 abandoned   sadness
##  9 abandonment anger
## 10 abandonment fear
## # ... with 13,891 more rows
```

```
example_words %>% inner_join(nrc, by = "word") %>% select(word, sentiment)
```

```
## # A tibble: 7 x 2
##   word  sentiment
##   <chr> <chr>
## 1 blue  sadness
## 2 sugar positive
## 3 sweet anticipation
## 4 sweet joy
## 5 sweet positive
## 6 sweet surprise
## 7 sweet trust
```

But this approach has many limitations. Each word is mapped to a sentiment and the word is prequalified in a dictionary. So a term like “this is not bad” is understood to have 2 negative sentiments instead of a positive (or at the least a neutral) sentiment. Synonyms, Sarcasm, colloquial word usage and different word usage are not directly considered.

An alternate approach is to wrangle the text data and use a dataset that has a sentiment marked against it and let the machine learning algorithm figure out the sentiment using the entire text directly. Lets see how this can be done.

Text mining with the tm package.

The tm package has great features to work with text data. It can read one or many documents and convert it into a *corpus* or a collection of documents. It can then transform this document to remove common words, punctuation, whitespaces etc to make it more easy to work with.

As an example, lets load the *Martin Luther King Speech* and work through the text mining process with this document.

```r
filePath <- "http://www.sthda.com/sthda/RDoc/example-files/martin-luther-king-i-have-a-dream-speech.txt
text <- readLines(filePath)
docs <- Corpus(VectorSource(text))

inspect(docs)
```

```
## <<SimpleCorpus>>
## Metadata: corpus specific: 1, document level (indexed): 0
## Content: documents: 46
##
## [1]
## [2] And so even though we face the difficulties of today and tomorrow, I still have a
dream. It is a dream deeply rooted in the American dream.
## [3]
## [4] I have a dream that one day this nation will rise up and live out the true meaning
of its creed:
## [5]
## [6] We hold these truths to be self-evident, that all men are created equal.
## [7]
## [8] I have a dream that one day on the red hills of Georgia, the sons of former slaves
and the sons of former slave owners will be able to sit down together at the table of
brotherhood.
## [9]
## [10] I have a dream that one day even the state of Mississippi, a state sweltering
with the heat of injustice, sweltering with the heat of oppression, will be transformed
into an oasis of freedom and justice.
## [11]
## [12] I have a dream that my four little children will one day live in a nation where
they will not be judged by the color of their skin but by the content of their character.
## [13]
## [14] I have a dream today!
## [15]
## [16] I have a dream that one day, down in Alabama, with its vicious racists, with its
governor having his lips dripping with the words of interposition and nullification, one
day right there in Alabama little black boys and black girls will be able to join hands
with little white boys and white girls as sisters and brothers.
## [17]
## [18] I have a dream today!
## [19]
## [20] I have a dream that one day every valley shall be exalted, and every hill and
mountain shall be made low, the rough places will be made plain, and the crooked places
will be made straight; and the glory of the Lord shall be revealed and all flesh shall
see it together.
## [21]
## [22] This is our hope, and this is the faith that I go back to the South with.
## [23]
## [24] With this faith, we will be able to hew out of the mountain of despair a stone of
hope. With this faith, we will be able to transform the jangling discords of our nation
into a beautiful symphony of brotherhood. With this faith, we will be able to work
together, to pray together, to struggle together, to go to jail together, to stand up for
freedom together, knowing that we will be free one day.
## [25]
## [26] And this will be the day, this will be the day when all of God s children will be
able to sing with new meaning:
## [27]
## [28] My country tis of thee, sweet land of liberty, of thee I sing.
## [29] Land where my fathers died, land of the Pilgrim s pride,
## [30] From every mountainside, let freedom ring!
## [31] And if America is to be a great nation, this must become true.
## [32] And so let freedom ring from the prodigious hilltops of New Hampshire.
## [33] Let freedom ring from the mighty mountains of New York.
## [34] Let freedom ring from the heightening Alleghenies of Pennsylvania.
## [35] Let freedom ring from the snow-capped Rockies of Colorado.
## [36] Let freedom ring from the curvaceous slopes of California.
## [37]
## [38] But not only that:
## [39] Let freedom ring from Stone Mountain of Georgia.
## [40] Let freedom ring from Lookout Mountain of Tennessee.
## [41] Let freedom ring from every hill and molehill of Mississippi.
## [42] From every mountainside, let freedom ring.
## [43] And when this happens, when we allow freedom ring, when we let it ring from every
village and every hamlet, from every state and every city, we will be able to speed up
that day when all of God s children, black men and white men, Jews and Gentiles,
Protestants and Catholics, will be able to join hands and sing in the words of the old
Negro spiritual:
## [44] Free at last! Free at last!
## [45]
## [46] Thank God Almighty, we are free at last!
```

```r
docs <- tm_map(docs, content_transformer(tolower))
docs <- tm_map(docs, removeNumbers)
docs <- tm_map(docs, removeWords, stopwords("english"))
docs <- tm_map(docs, removeWords, c("Georgia", "Mississippi"))
docs <- tm_map(docs, removePunctuation)
docs <- tm_map(docs, stripWhitespace)

getTransformations()
```

```
## [1] "removeNumbers"     "removePunctuation" "removeWords"
## [4] "stemDocument"      "stripWhitespace"
```

It can additionall stem the document - which essentially normalizes all words with the same root. So for example, likes, liked, likely, liking are all transformed as *like*. Additionally we can convert the document into a DocumentTermMatrix using which we can find word frequencies and even build a word cloud.

```
writeLines(as.character(docs[[2]]))
```

```
##  even though face difficulties today tomorrow still dream dream deeply rooted american dream
```

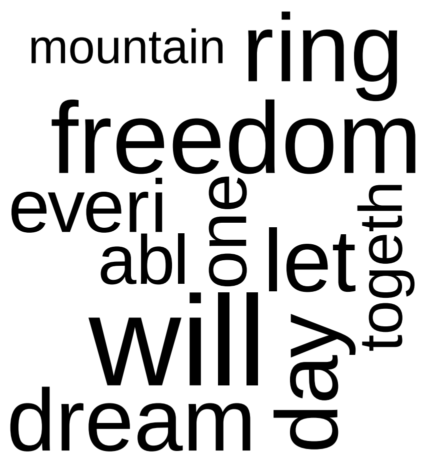
```
docs <- tm_map(docs, stemDocument)
writeLines(as.character(docs[[2]]))
```

```
## even though face difficulti today tomorrow still dream dream deepli root american dream
```

```
dcm <- DocumentTermMatrix(docs)
freq <- findFreqTerms(dcm, lowfreq = 5)
freq
```

```
##  [1] "dream"    "day"      "one"      "will"     "abl"      "togeth"
##  [7] "freedom"  "everi"    "mountain" "let"      "ring"
```

```
tfreq <- colSums(as.matrix(dcm))
wordcloud(names(tfreq),tfreq, min.freq=5)
```



This allows us to wrangle the data on an existing sentiment dataset and use algorithms like linear regression or knn to improve our sentiment predictions.

Analysis and Results

Let's use a sample dataset of Tweets about Apple from Kaggle from https://www.kaggle.com/c/apple-computers-twitter-sentiment2/data (Note:This requires a login and so the train.csv has been copied to a github repo. You can just download the train.csv from the kaggle site and replace the one that is being used).

```
#Set up so that both R and RStudio share the same working directory.
#Change this appropriately on Mac/Linux or just remove altogether
setwd("c:\\temp")
dl <- "train.csv"

# Check for download. If exists, just load it, else download, process and save it.
if(!file.exists(dl))
{
  download.file("https://raw.githubusercontent.com/srivatsapraveen/HDXCapstone/master/02_SentimentAnaly
}

tweets <- read.csv(dl, stringsAsFactors = FALSE)
tweets %>% group_by(sentiment) %>% summarize(n())
```

```
## # A tibble: 3 x 2
##   sentiment `n()`
##       <int> <int>
## 1         1   608
## 2         3  1061
## 3         5   218
```

As we can see, this has a set of tweets along with a basic score of 1-Negative, 3-Neutral and 5-Positive. For our consideration, lets just map both 1 and 3 to a "N"-Negative score and 5 to a "P"-Positive score. We can group this by the sentiment to see how our tweets behave.

```
tweets <- tweets %>% mutate(senti_text = ifelse(sentiment > 3,"P","N"))
tweets$senti_text <- as.factor(tweets$senti_text)
tibble(tweets)
```

```
## # A tibble: 1,887 x 1
##    tweets$sentiment $text                                            $senti_text
##               <int> <chr>                                            <fct>
##  1                5 "RT @tomlindberg99 'Congrats to @Apple, Mary So... P
##  2                3 "Pop up #BostonHoliday - #UrbanNutcracker's @Ru... N
##  3                3 "New 4-inch iPhone 6 Reported. 5 Reasons Apple ... N
##  4                1 "@Apple GBP200 repair an #iPhone5 that died 4 m... N
##  5                3 "Foreign Currency Exchange May Be A Headwind Fo... N
##  6                1 "Mark #Zuckerberg Slams Tim #Cook: You Don't Ca... N
##  7                5 "@AppleOfficialll @apple I can't say enough abo... P
##  8                3 "@SamsungMobileUS #Samsung #appeals $930m fine ... N
##  9                3 "BLOCK TRADE detected in #AAPL"                  N
## 10                3 "Is Apple Inc. Losing the $5 Billion Ed-Tech Ma... N
## # ... with 1,877 more rows
```

```
tweets %>% group_by(senti_text) %>% summarize(n())
```

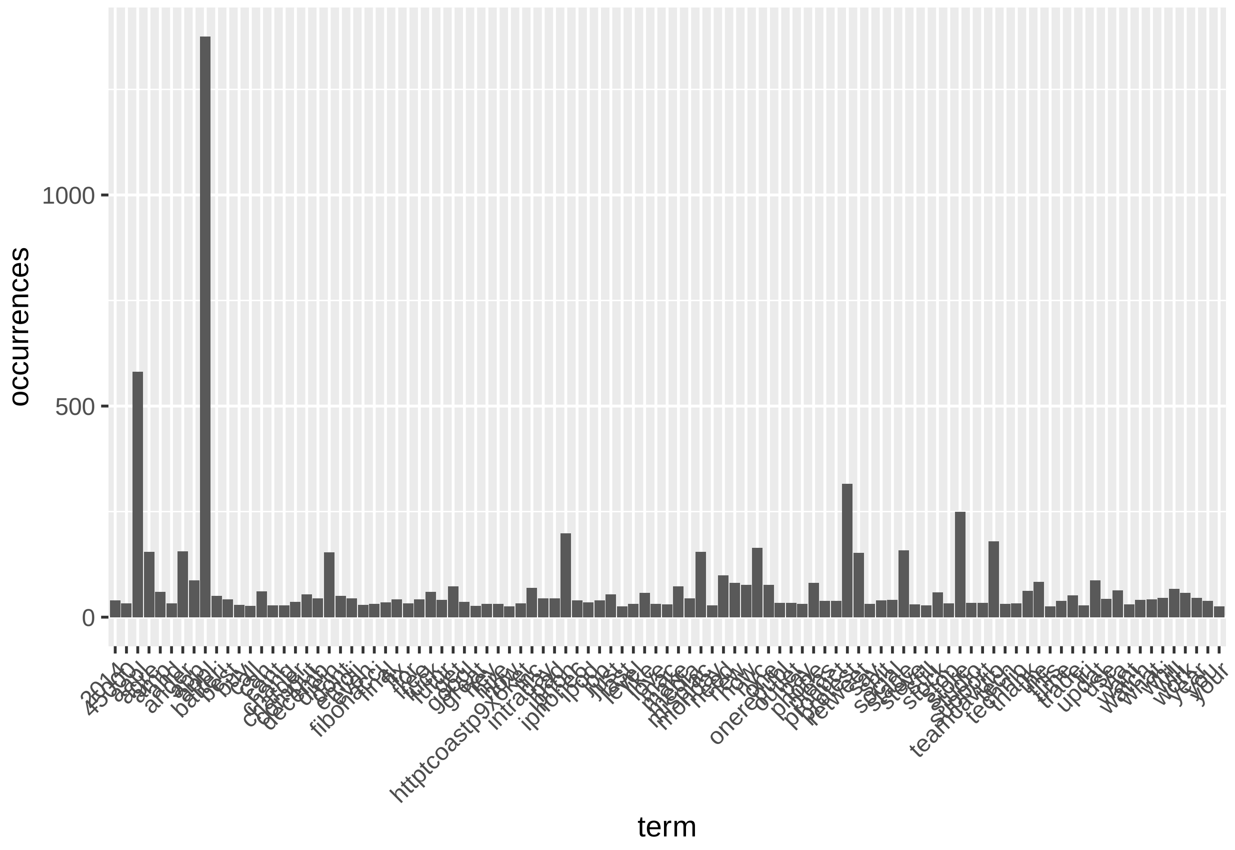
```
## # A tibble: 2 x 2
##   senti_text `n()`
##   <fct>      <int>
## 1 N           1669
## 2 P            218
```

Lets then cleanup the text using the tm package and the transformations that we saw earlier.

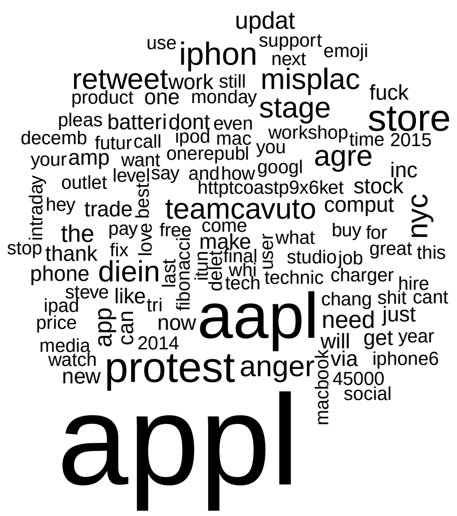
```
tweet_docs <- VCorpus(VectorSource(tweets$text))
tweet_docs <- tm_map(tweet_docs, PlainTextDocument)
tweet_docs <- tm_map(tweet_docs, removePunctuation)
tweet_docs <- tm_map(tweet_docs, removeWords, c("apple", stopwords("english")))
tweet_docs <- tm_map(tweet_docs, content_transformer(tolower))
tweet_docs <- tm_map(tweet_docs, stemDocument)

tweet_docMatrix <- DocumentTermMatrix(tweet_docs)
wordfreq <- findFreqTerms(tweet_docMatrix, lowfreq = 25)

totfreq <- colSums(as.matrix(tweet_docMatrix))
freqword=data.frame(term=names(totfreq),occurrences=totfreq)
p <- ggplot(subset(freqword, totfreq>25), aes(term, occurrences))
p <- p + geom_bar(stat="identity")
p <- p + theme(axis.text.x=element_text(angle=45, hjust=1))
p
```



```r
#setting the same seed each time ensures consistent look across clouds
set.seed(123, sample.kind="Rounding")
#limit words by specifying min frequency
wordcloud(names(totfreq),totfreq, min.freq=25)
```



Now let us use this transpose the words in the twitter text into columns so that each row can be analyzed for each of the frequently used words. This enables us to create a co-relation across words and map the same to the reported sentiment.

```r
freqDocs <- removeSparseTerms(tweet_docMatrix, 0.995)
freqTweets <- as.data.frame(as.matrix(freqDocs))
colnames(freqTweets) <- make.names(colnames(freqTweets))

freqTweets$sentiment <- tweets$sentiment
freqTweets$senti_text <- tweets$senti_text
tibble(freqTweets)
```

```
## # A tibble: 1,887 x 1
##    freqTweets$X2014 $X2015 $X45000 $X4inch $X500 $aapl $aaplappl $actual $agre
##               <dbl>  <dbl>   <dbl>   <dbl> <dbl> <dbl>     <dbl>   <dbl> <dbl>
##  1                0      0       0       0     0     0         0       0     0
##  2                0      0       0       0     0     0         0       0     0
##  3                0      0       0       1     0     2         0       0     0
##  4                0      0       0       0     0     0         0       0     0
##  5                0      0       0       0     0     1         0       0     0
##  6                0      0       0       0     0     1         0       0     0
##  7                0      0       0       0     0     0         0       0     0
##  8                0      0       0       0     0     0         0       0     0
##  9                0      0       0       0     0     1         0       0     0
## 10                0      0       0       0     0     1         0       0     0
## # ... with 1,877 more rows, and 301 more variables: $air <dbl>, $album <dbl>,
## #   $all <dbl>, $amazon <dbl>, $amp <dbl>, $and <dbl>, $android <dbl>,
## #   $anger <dbl>, $antitrust <dbl>, $app <dbl>, $appl <dbl>, $are <dbl>,
## #   $autocorrect <dbl>, $awesom <dbl>, $back <dbl>, $bad <dbl>, $batteri <dbl>,
## #   $best <dbl>, $better <dbl>, $big <dbl>, $block <dbl>, $blog <dbl>,
## #   $brand <dbl>, $build <dbl>, $buy <dbl>, $call <dbl>, $camera <dbl>,
## #   $can <dbl>, $cant <dbl>, $case <dbl>, $center <dbl>, $ceo <dbl>,
## #   $chang <dbl>, $charg <dbl>, $charger <dbl>, $china <dbl>, $code <dbl>,
## #   $come <dbl>, $comment <dbl>, $compani <dbl>, $comput <dbl>, $cook <dbl>,
## #   $could <dbl>, $creat <dbl>, $cruci0btch <dbl>, $custom <dbl>, $day <dbl>,
## #   $dear <dbl>, $decemb <dbl>, $delet <dbl>, $demand <dbl>, $detect <dbl>,
## #   $develop <dbl>, $devic <dbl>, $die <dbl>, $diein <dbl>, $differ <dbl>,
## #   $doesnt <dbl>, $dont <dbl>, $drop <dbl>, $earn <dbl>, $elev <dbl>,
## #   $emoji <dbl>, $even <dbl>, $ever <dbl>, $everi <dbl>, $excit <dbl>,
## #   $face <dbl>, $facebook <dbl>, $facetim <dbl>, $fashion <dbl>,
## #   $featur <dbl>, $fibonacci <dbl>, $final <dbl>, $first <dbl>, $fix <dbl>,
## #   $follow <dbl>, $fool <dbl>, $for. <dbl>, $free <dbl>, $friday <dbl>,
## #   $from <dbl>, $fuck <dbl>, $futur <dbl>, $game <dbl>, $garag <dbl>,
## #   $gay <dbl>, $genius <dbl>, $get <dbl>, $good <dbl>, $googl <dbl>,
## #   $got <dbl>, $great <dbl>, $guru <dbl>, $guy <dbl>, $happen <dbl>,
## #   $harri <dbl>, $hate <dbl>, $have <dbl>, $help <dbl>, ...
```

```r
nrow(freqTweets)
```

```
## [1] 1887
```

```r
ncol(freqTweets)
```

```
## [1] 310
```

This dataset now has 310 columns for each of the 1887 rows and it has a count of the number of occurences of the word in that row. This matrix now lends itself to be used for predicting the sentiment - "P"-Positive or "N"-Negative for any tweet or short text that gets posted.

Regression Model to predict the twitter sentiment.

Lets first start with a regression model to predict the sentiment. As we do for all machine learning models, we partition the dataset into a train and test set. We use the train set to train the model and then we evaluate or predict the outcome based on the test set. We then use the accuracy from the confusionMatrix function of the caret package to compare how our algorithms behaved.

```r
set.seed(123, sample.kind="Rounding")
test_index <- createDataPartition(freqTweets$senti_text, times = 1, p = 0.7, list = FALSE)
train_set <- freqTweets %>% slice(-test_index)
test_set <- freqTweets %>% slice(test_index)

lm_fit <- mutate(train_set, y = as.numeric(senti_text == "P")) %>% lm(y ~ ., data = .)
p_hat <- predict(lm_fit, test_set)
y_hat <- ifelse(p_hat == 1, "P", "N") %>% factor()
confusionMatrix(y_hat, test_set$senti_text)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    N    P
##          N 1169  110
##          P    0   43
##
##                Accuracy : 0.9168
##                  95% CI : (0.9006, 0.9311)
##     No Information Rate : 0.8843
##     P-Value [Acc > NIR] : 6.94e-05
##
##                   Kappa : 0.4088
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.2810
##          Pos Pred Value : 0.9140
##          Neg Pred Value : 1.0000
##              Prevalence : 0.8843
##          Detection Rate : 0.8843
##    Detection Prevalence : 0.9675
##       Balanced Accuracy : 0.6405
##
##        'Positive' Class : N
##
```

```r
confusionMatrix(y_hat, test_set$senti_text)$overall[["Accuracy"]]
```

```
## [1] 0.9167927
```

So using a simple regression model, we have been able to predict the sentiment with a 0.9168 accuracy. Now, lets explore another algorithms - the K-Nearest Neighbour (KNN) and see how it behaves.

```r
train_knn <- train(senti_text ~ ., method = "knn", data = train_set)
y_hat_knn <- predict(train_knn, test_set, type = "raw")
confusionMatrix(y_hat_knn, test_set$senti_text)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    N    P
##          N 1169   10
##          P    0  143
##
##                Accuracy : 0.9924
##                  95% CI : (0.9861, 0.9964)
##     No Information Rate : 0.8843
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.962
```

```
##
##  Mcnemar's Test P-Value : 0.004427
##
##             Sensitivity : 1.0000
##             Specificity : 0.9346
##          Pos Pred Value : 0.9915
##          Neg Pred Value : 1.0000
##              Prevalence : 0.8843
##          Detection Rate : 0.8843
##    Detection Prevalence : 0.8918
##       Balanced Accuracy : 0.9673
##
##        'Positive' Class : N
##
```

```
confusionMatrix(y_hat_knn, test_set$senti_text)$overall[["Accuracy"]]
```

```
## [1] 0.9924357
```

The KNN model give us an accuracy of 0.9924 This makes sense as the KNN model creates a nearness algorithms between different words that contribute to the sentiment instead of just taking a look at one word at a time giving us a higher accuracy. Lets also explore the random forest algrithm.

```
train_RF <- randomForest(senti_text ~ . , data = train_set)
predict_RF <- predict(train_RF, newdata = test_set)
confusionMatrix(predict_RF, test_set$senti_text)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    N    P
##          N 1165    0
##          P    4  153
##
##                Accuracy : 0.997
##                  95% CI : (0.9923, 0.9992)
##     No Information Rate : 0.8843
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9854
##
##  Mcnemar's Test P-Value : 0.1336
##
##             Sensitivity : 0.9966
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9745
##              Prevalence : 0.8843
##          Detection Rate : 0.8812
##    Detection Prevalence : 0.8812
##       Balanced Accuracy : 0.9983
##
##        'Positive' Class : N
##
```

```r
confusionMatrix(predict_RF, test_set$senti_text)$overall["Accuracy"]
```

```
##  Accuracy
## 0.9969743
```

The random forest gives us an accuracy of 0.9969. Keep in mind that the dataset is small and within this limited set we have a larger set of negative sentiment as compared to the positive sentiment. But even with this, we now have an approach where we can use different algorithms to compare the sentiment across various social network postings and identifying a sentiment for a product or service.

Summary and Conclusion

Sentiment analysis using text data from social media like twitter and comments and posts from different sites is a great way to glean user sentiment from a broad worldwide audience in near real time. While a basic sentiment analysis can be done using a pre-defined dictionary, using datasets and applying machine learning algorithms are more effective is providing better accuracy for our sentiment predictions.

However, this project continues to have many *limitations*. To begin with the dataset was small and very specific to apple products. As the number of comments increases, the number of words also increases. This can dramatically increase the sparse matrix and can make the computation very heavy. While other limitations like ability to detect double negatives ("not bad") is better, the algorithm is still not very effective at detecting sarcasm.

Going forward, as part of the *future work*, the text mapping can be fine tuned. For example, words appearing in the same sentence have more weight than ones that appear across sentences. The sequence of words ("not bad", vs "bad, not") can also be mapped for a better co-relation between text construction and the sentiment. Lastly, people across the globe use different writing styles and new media platforms like twitter encourages short texts (nb - for not bad) or the use of emojis. This makes identifying sentiment from such posts much more challenging. All these can be taken into account to further fine tune the sentiment prediction based on text comments.

References and Citation

1. https://rafalab.github.io/dsbook/text-mining.html#sentiment-analysis
2. http://www.sthda.com/english/wiki/text-mining-and-word-cloud-fundamentals-in-r-5-simple-steps-you-should-know
3. https://eight2late.wordpress.com/2015/05/27/a-gentle-introduction-to-text-mining-using-r/