Text Analytics- Sentiment Analytics of Twitter Data

Karishma Yadav

6/26/2020

The goal of this project is to project the sentiment of a tweet using Bag of Words method. For this, we first load the dataset tweet.csv into R and pre-process it. Since we are dealing with text data here, so we set the stringsAsFactors argument as FALSE.

```
tweets <- read.csv("tweets.csv", stringsAsFactors = FALSE)
str(tweets)
```

```
## 'data.frame':    1181 obs. of  2 variables:
##  $ Tweet: chr  "I have to say, Apple has by far the best customer care service I have ever received!
##  $ Avg  : num  2 2 1.8 1.8 1.8 1.8 1.8 1.6 1.6 1.6 ...
```

We have 1181 observations for 2 variables:

Tweet: the text of the real tweets directed to the company Apple Avg: the average sentiment score of the tweets

We are more interested in finding the negative sentiment, so we process our data by setting the negative tweets as TRUE if the average sentiment of the tweet is less than -1, as follows:

```
tweets$Negative <- as.factor(tweets$Avg <= -1)
table(tweets$Negative)
```

```
##
## FALSE  TRUE
##   999   182
```

Now to use the Bag of Words approach, we need to pre-process our data using the Text Mining packages. We need to convert our text into documents to pre-process it:

```
library(tm)
```

```
## Loading required package: NLP
```

```
library(SnowballC)
corpus <- Corpus(VectorSource(tweets$Tweet))
corpus
```

```
## <<SimpleCorpus>>
## Metadata:  corpus specific: 1, document level (indexed): 0
## Content:  documents: 1181
```

```
corpus[[1]]
```

```
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 101
```

```
corpus <- tm_map(corpus, tolower)
```

```
## Warning in tm_map.SimpleCorpus(corpus, tolower): transformation drops documents
```

```
corpus <- tm_map(corpus, removePunctuation)
```

```
## Warning in tm_map.SimpleCorpus(corpus, removePunctuation): transformation drops
## documents
```

```
corpus <- tm_map(corpus, removeWords, c("apple", stopwords("english")))
```

```
## Warning in tm_map.SimpleCorpus(corpus, removeWords, c("apple",
## stopwords("english"))): transformation drops documents
```

```
corpus <- tm_map(corpus, stemDocument)
```

```
## Warning in tm_map.SimpleCorpus(corpus, stemDocument): transformation drops
## documents
```

```
corpus[[1]]
```

```
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 51
```

We have pre processed our data and we are ready to generate word frequencies to use in our prediction model. We use DocumentTermMatrix function which gives us the value of number of times a word is used in each document.

```
frequencies <- DocumentTermMatrix(corpus)
frequencies
```

```
## <<DocumentTermMatrix (documents: 1181, terms: 3289)>>
## Non-/sparse entries: 8980/3875329
## Sparsity           : 100%
## Maximal term length: 115
## Weighting          : term frequency (tf)
```

We see that we have 3289 terms in 1181 documents, let us see what this matrix looks like using the inspect function. And check the terms with frequency 20 or more.

```
inspect(frequencies[1000:1005, 500:515])
```

```
## <<DocumentTermMatrix (documents: 6, terms: 16)>>
## Non-/sparse entries: 1/95
## Sparsity           : 99%
## Maximal term length: 23
## Weighting          : term frequency (tf)
## Sample             :
##       Terms
## Docs   asap dead follow idea lion mountain save ssd take touchid
##   1000    0    0      0    1    0        0    0   0    0       0
##   1001    0    0      0    0    0        0    0   0    0       0
##   1002    0    0      0    0    0        0    0   0    0       0
##   1003    0    0      0    0    0        0    0   0    0       0
##   1004    0    0      0    0    0        0    0   0    0       0
##   1005    0    0      0    0    0        0    0   0    0       0
```

```
findFreqTerms(frequencies, lowfreq = 20)
```

```
##  [1] "say"                 "love"                "iphon"
##  [4] "iphone5"             "new"                 "thank"
##  [7] "phone"               "can"                 "make"
## [10] "market"              "one"                 "will"
## [13] "cant"                "get"                 "just"
## [16] "updat"               "fingerprint"         "iphone5c"
## [19] "store"               "time"                "come"
## [22] "now"                 "use"                 "back"
## [25] "anyon"               "work"                "app"
## [28] "android"             "think"               "ipad"
## [31] "well"                "freak"               "dont"
## [34] "via"                 "better"              "like"
## [37] "pleas"               "samsung"             "want"
## [40] "batteri"             "ios7"                "microsoft"
## [43] "itun"                "buy"                 "releas"
## [46] "look"                "appl"                "need"
## [49] "googl"               "twitter"             "ipod"
## [52] "ipodplayerpromo"     "promoipodplayerpromo" "lol"
## [55] "realli"              "promo"
```

We still have 56 terms, which is too much for our prediction model and may lead to overfitting. Let us reduce the number of tweets further using the removeSparseTerms function by keeping the sparsity threshold as 0.995, this will keep the terms which appear in 0.5% of more of the tweets.

```
sparse <- removeSparseTerms(frequencies, 0.995)
sparse
```

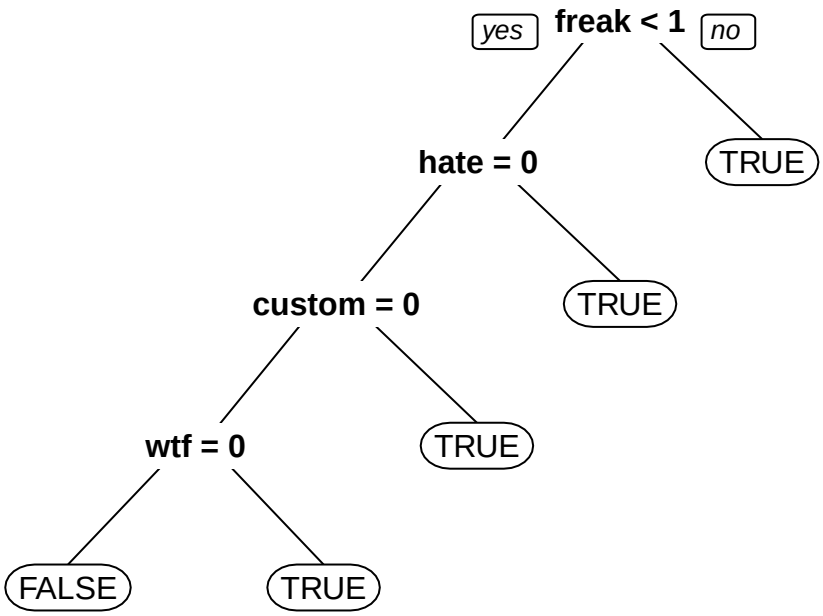
```
## <<DocumentTermMatrix (documents: 1181, terms: 309)>>
## Non-/sparse entries: 4669/360260
## Sparsity           : 99%
## Maximal term length: 20
## Weighting          : term frequency (tf)
```

Now we will convert our sparse matrix to a dataframe to be used in our model.

```
tweetsSparse <- as.data.frame(as.matrix(sparse))
colnames(tweetsSparse) <- make.names(colnames(tweetsSparse))
tweetsSparse$Negative = tweets$Negative
```

Now let us build the model:

```
library(caTools)
library(rpart)
library(rpart.plot)
split <- sample.split(tweetsSparse$Negative, SplitRatio =0.7 )
trainsparse <- subset(tweetsSparse, split==TRUE)
testsparse <- subset(tweetsSparse, split==FALSE)
tweetCart <- rpart(Negative~., data=trainsparse, method="class")
prp(tweetCart)
```



Our tree says that if the word freak is in the tweet then the sentiment is Negative, if the word freak is not in the tweet but the word stuff is, again Negative sentiment. And if neither of the two is there in the tweet but the word hate is, then also the sentiment is negative. If none of these 3 words are there in the tweet then predict non negative sentiment.

Now let us predict our data:

```
predictCart <- predict(tweetCart, newdata = testsparse, type="class")
#Building the confusion matrix:
table(testsparse$Negative, predictCart)
```

```
##        predictCart
##         FALSE TRUE
##   FALSE   294    6
##   TRUE     35   20
```

Accuracy of the Model:

```
(294+19)/(296+4+36+19)
```

```
## [1] 0.8816901
```

```
table(testsparse$Negative)
```

```
##
## FALSE  TRUE
##   300    55
```

This tells us that the accuracy of our model is 0.88, which is pretty good. By tabling the baseline model, we see that 300 tweets have non-negative sentiment and 55 tweets with negative sentiment. The accuracy of the baseline model is 300/355=0.85, which means that our cart model does better than this simple baseline model.

Let us see how well a random forest model will do.

```
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
tweetRF <- randomForest(Negative ~., data=trainsparse)
predictRF <- predict(tweetRF, newdata=testsparse)
table(testsparse$Negative, predictRF)
```

```
##        predictRF
##         FALSE TRUE
##   FALSE   294    6
##   TRUE     29   26
```

The accuracy of the random Forest model is

```
(294+21)/(294+21+6+34)
```

```
## [1] 0.8873239
```

This is a little bit more than the accuracy of the cart model. But the interpretability of the cart model is a bit better than the random forest model.