Sentiment Analysis using Machine Learning in R

Meghana RV

26/12/2021

Contents

1 Abstract

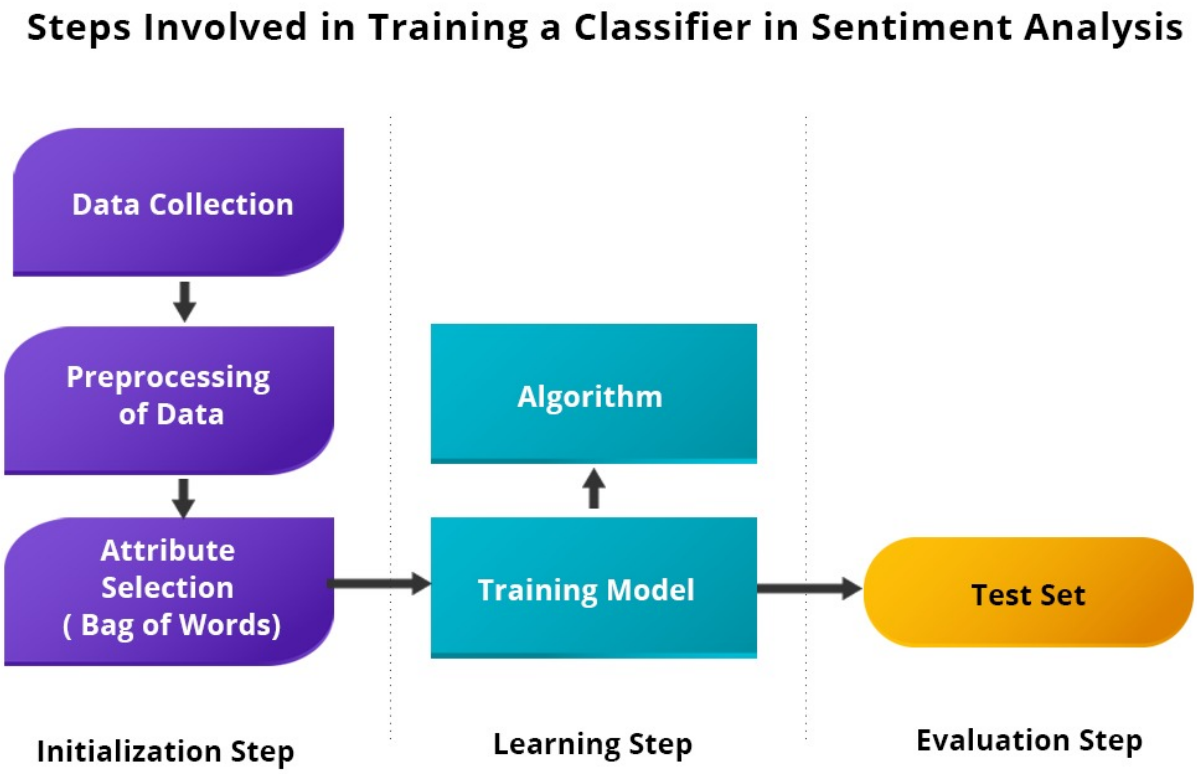
This project is the final project, part of the HarvardX : PH125.9x Data Science : Capstone course. In this project, we train a machine learning algorithm to perform Sentiment Analysis, to decide whether we can or cannot recommend a chosen product from Amazon to a new customer. Sentiment Analysis is used to gauge the 'sentiment' or 'emotion' that is felt by a person with regard to a certain entity. These 'entities' include, but are not restricted to products from online shopping websites, public tweets, views on politics, other opinions, etc. 'Sentiment' can be positive, negative or neutral. Here, we choose a product from Amazon and attempt to measure the sentiment associated with it. If the proportion of positive sentiment is a lot larger than negative sentiment, we can say that it is has good reviews and is a reliable product. It can thus be recommended to a new customer who is interested in purchasing it. Similarly, if the proportion of negative sentiment is too high (too many bad reviews) then we don't recommend it.

The data for this project was obtained from Kaggle in the form of a csv file. It contains 5000 reviews of various products, (about 25) , that were posted by Amazon-customers, on Amazon. Two main approaches were implemented to perform Sentiment Analysis : Rule or Lexicon based Approach and the Naive Bayes Approach. The accuracy of both models is calculated. In the end we decide whether the selected product can or cannot be recommended.

2 Introduction

Classification algorithms in machine learning use input training data to predict the likelihood that subsequent data will fall into one of the predetermined categories. One of the most common uses of classification is filtering emails into "spam" or "non-spam." In short, classification is a form of "pattern recognition," with classification algorithms applied to the training data to find the same pattern (similar words or sentiments, number sequences, etc.) in future sets of data.

Sentiment analysis is a machine learning tool that analyzes texts for polarity, from positive to negative. By training machine learning tools with examples of emotions in text, machines automatically learn how to detect sentiment without human input. Sentiment analysis is predominantly a classification algorithm, aimed at finding an opinionated point of view and its disposition and highlighting the information of particular interest in the process.



There are roughly three algorithm models for sentiment analysis :

1. **Rule or Lexicon Based Approach :**

   This approach relies on manually crafted rules for data classification to determine sentiment. This approach use dictionaries of words with positive or negative values to denote their polarity and sentiment strength to calculate a score. Additional functionality can also be added by including expressions. Rule based sentiment analysis algorithms can be customized based on context by developing even smarter rules.

2. **Automated or Machine Learning approach :**

   Instead of clearly defined rules, this sentiment analysis model uses machine learning to figure out the essence of the statement. This ensures that the exactitude of the analysis improves and information

can be processed on many criteria without it being too complicated. This approach involves the use of machine learning algorithms under supervision. An algorithm is trained with many sample passages until it can predict with accuracy the sentiment of the text. Then large pieces of text are fed into the classifier and it predicts the sentiment as negative, neutral or positive.

Machine learning models can be of two kinds :

i.Traditional Models : This method requires the gathering of a dataset with examples for positive, negative, and neutral classes, then processing this data, and finally training the algorithm based on the examples. These methods are mainly used for determining the polarity of text.

Traditional machine learning methods such as Naive Bayes, Logistic Regression and Support Vector Machines (SVM) are widely used for large-scale sentiment analysis because they are capable of scalability.

ii.Deep Learning Models : This provides more precise results than traditional models and includes neural network models such as CNN (Convoluted Neural Network), RNN (Recurrent Neural Network), and DNN (Deep Neural Network).

The main models used for sentiment analysis classification algorithms are Naive Bayes and Deep Learning.

3. **Hybrid Approach :**

   Hybrid sentiment analysis models are the most modern, efficient, and widely-used approach for sentiment analysis. Provided you have well-designed hybrid systems, you can actually get the benefits of both automatic and rule-based systems. Hybrid models can offer the power of machine learning coupled with the flexibility of customization.

The choice of the approach used would vary depending on the application.

3 Methods / Analysis of the Data

We initially start off with a csv file called '5000 REVIEWS'. As mentioned before, this file contains 5000 reviews from various customers, about various products.

It was found that the product with second highest number of reviews (590 reviews) is :

Amazon - Echo Plus w/ Built-In Hub - Silver.

Compared to another product with nearly 900 reviews, this was chosen to ease pre-processing. Some manual processing was required, which would have been very difficult to implement with 900 reviews.

The following files were created, edited and made use of, during analysis. The use of each file will be understood later.

Table 1: Description of the files used during analysis

Name of File	**Type**	**Description**
5000 REVIEWS	csv	Original file. Contains all 5000 reviews with other unnecessary info like id-number, manufacturer, date of review, etc.
Reviews	csv	Has all 5000 reviews with only relevant info (name of product, reviews and the reviews.doRecommend* column)
Reviews2	csv	Has all the reviews associated with the chosen product : *Amazon - Echo Plus w/ Built-In Hub - Silver* and the reviews.doRecommend column (there are 590 reviews)
Reviews2 - Copy	csv	Contains only the 590 reviews of *Amazon - Echo Plus w/ Built-In Hub - Silver*
Reviews2 - Copy_1	csv	Reviews2 - Copy was edited such that there are only two cells. One containing the product name. The other contains all reviews combined as a paragraph. This is saved as Reviews2 - Copy_1.csv
new	xlsx	Any and all types of symbols are removed. The words from the 590 reviews are saved such that nearly 15 words make up a single cell in the Excel sheet
data_scr	csv	Each line is assigned a score of 0 or 1. If the review is positive, that cell is assigned a score of 1. If the review is negative the corresponding cell is assigned a score of 0. (This was done manually)

** reviews.doRecommend is a column that indicates whether the a review recommends the product or not. Each review is assigned either 'TRUE' or 'FALSE', depending on whether the review is positive or negative respectively.*

Pre-processing

The first step is to load the required packages.

The working directory must be the same as the location of the **5000 REVIEWS.csv** file. If not, when asked to import it, the console window would throw an error, since it would be unable to access the file.

```
# set working directory
setwd("C:/Users/MAG/Documents/CYO")
```

Now we can import it.

```
dat <- read_csv("5000 REVIEWS.csv")
```

The raw data looks like this :

```
## # A tibble: 6 x 24
##   id     dateAdded           dateUpdated         name    asins brand categories
##   <chr>  <dttm>              <dttm>              <chr>   <chr> <chr> <chr>
## 1 AVqVG~ 2017-03-03 16:56:05 2018-10-25 16:36:31 "Amazo~ B00Z~ Amaz~ Computers,~
## 2 AVqVG~ 2017-03-03 16:56:05 2018-10-25 16:36:31 "Amazo~ B00Z~ Amaz~ Computers,~
## 3 AVqVG~ 2017-03-03 16:56:05 2018-10-25 16:36:31 "Amazo~ B00Z~ Amaz~ Computers,~
## 4 AVqVG~ 2017-03-03 16:56:05 2018-10-25 16:36:31 "Amazo~ B00Z~ Amaz~ Computers,~
## 5 AVqVG~ 2017-03-03 16:56:05 2018-10-25 16:36:31 "Amazo~ B00Z~ Amaz~ Computers,~
## 6 AVqVG~ 2017-03-03 16:56:05 2018-10-25 16:36:31 "Amazo~ B00Z~ Amaz~ Computers,~
## # ... with 17 more variables: primaryCategories <chr>, imageURLs <chr>,
## #   keys <chr>, manufacturer <chr>, manufacturerNumber <chr>,
## #   reviews.date <dttm>, reviews.dateAdded <dttm>, reviews.dateSeen <chr>,
## #   reviews.doRecommend <lgl>, reviews.id <dbl>, reviews.numHelpful <dbl>,
## #   reviews.rating <dbl>, reviews.sourceURLs <chr>, reviews.text <chr>,
## #   reviews.title <chr>, reviews.username <chr>, sourceURLs <chr>
```

The output of pre-processing is saved as a different file, since we want to retain the original file.

```
new_dat <- select(dat, -c(id, asins, categories, dateAdded, dateUpdated, brand,
                          imageURLs, keys, manufacturer, manufacturerNumber,
                          reviews.date, reviews.dateAdded, reviews.dateSeen,
                          reviews.id, reviews.numHelpful, reviews.rating,
                          reviews.sourceURLs, reviews.username, sourceURLs,
                          primaryCategories))

write.csv(new_dat, file = "Reviews.csv")
```

It looks like this :

```
## # A tibble: 6 x 4
##   name            reviews.doRecomm~ reviews.text              reviews.title
##   <chr>           <lgl>             <chr>                     <chr>
## 1 "Amazon Kindle~ FALSE             I thought it would be as~ Too small
## 2 "Amazon Kindle~ TRUE              This kindle is light and~ Great light reade~
## 3 "Amazon Kindle~ TRUE              Didnt know how much i'd ~ Great for the pri~
## 4 "Amazon Kindle~ TRUE              I am 100 happy with my p~ A Great Buy
## 5 "Amazon Kindle~ TRUE              Solid entry level Kindle~ Solid entry-level~
## 6 "Amazon Kindle~ FALSE             This make an excellent e~ Good ebook
```

Next we create Reviews2 - Copy.csv .

```
dat_final <- new_dat %>% filter(name == "Amazon - Echo Plus w/ Built-In Hub - Silver")

# extracting the reviews
para <- dat_final %>% group_by(name, reviews.text) %>%
  summarise_all(funs(trimws(paste(., collapse = ''))))

para1 <- para %>% select(reviews.text)

# save it as a file
write.csv(para1, file = "Reviews2 - Copy.csv")
```

In our first model, we follow the Lexicon based approach. It requires all the words in a 'Tokenized' format. Tokenization is the **process of splitting a string into a list of tokens.** That is, each word in a sentence is saved as an individual object in a list. Each word is called a *token.* The output is a list containing two columns - the first is just an index. The second contains a single word.

It can be understood by looking at the output of the code below :

First we create a vector with all 590 reviews. The tokens will be derived from this vector.

Showing the code that creates the text will make the report appear very cluttered. Hence the output alone is shown below. (To get an idea, when pasting the text in MS Word, it occupies nearly 7 pages.)

Before Tokenizing, we will convert it into a tibble.

```
text_df <- tibble(text = text)
```

It looks like this :

```
## # A tibble: 6 x 1
##   text
##   <chr>
## 1 A fun little device on the cutting edge of consumer artificial intelligence S~
## 2 pretty good Alexa does have some problems understanding how I ask questions a~
## 3 amount of the time A great value and I love it sounds great and works A must ~
## 4 should get one A perfect gift for someone who has everything and a busy sched~
## 5 use of creativity and design where installation was a breeze and use was pure~
## 6 Alexa hub and expanded the lighting capabilities Easy setup and it works like~
```

```
# Tokensizing
text_df <- text_df %>%
  unnest_tokens(word, text)
```

Here's what tokens look like :

```
head(text_df)
```

```
## # A tibble: 6 x 1
##   word
##   <chr>
## 1 a
## 2 fun
```

```
## 3 little
## 4 device
## 5 on
## 6 the
```

Now we can implement the first model.

Model 1

Lexicon based approach

A lexicon is the vocabulary of a language or branch of knowledge. In order to build our project on sentiment analysis, we will make use of the `tidytext` package that comprises of sentiment lexicons that are present in the dataset `sentiments`.

These three lexicons make use of **unigrams**. A unigram is a one-word sentence. These words are derived from textual data (in our case, the object `text`). Each word the assigned a score based on the sentiment that it denotes.

In general, there are three general purpose lexicons :

1. **AFFIN :** Words are scored in the range -5 to +5. An increase in negativity corresponds to negative sentiment whereas an increase in positivity corresponds to a positive one.
2. **Bing :** The bing lexicon model on the other hand, classifies sentiments into a binary category of negative (0) or positive (1).
3. **Loughran :** This model is used in financial market analysis.

We will make use of the bing lexicons to extract the sentiments out of our data. This is done using the `get_sentiments()` function.

Here is a sample of the data in the `get_sentiments()` function :

```
## # A tibble: 6 x 2
##   word       sentiment
##   <chr>      <chr>
## 1 2-faces    negative
## 2 abnormal   negative
## 3 abolish    negative
## 4 abominable negative
## 5 abominably negative
## 6 abominate  negative
```

```
#implement filter() over the words that correspond to positivity
positive_senti <- get_sentiments("bing") %>%
  filter(sentiment == "positive")

# how many positive words and their word count
text_df %>%
    semi_join(positive_senti) %>%
  count(word, sort = TRUE)
```

```
## # A tibble: 96 x 2
##    word        n
##    <chr>   <int>
##  1 great      71
##  2 love       59
##  3 easy       51
##  4 smart      43
##  5 like       22
```

```
##  6 works      22
##  7 fun        18
##  8 good       18
##  9 well       14
## 10 awesome    11
## # ... with 86 more rows
```

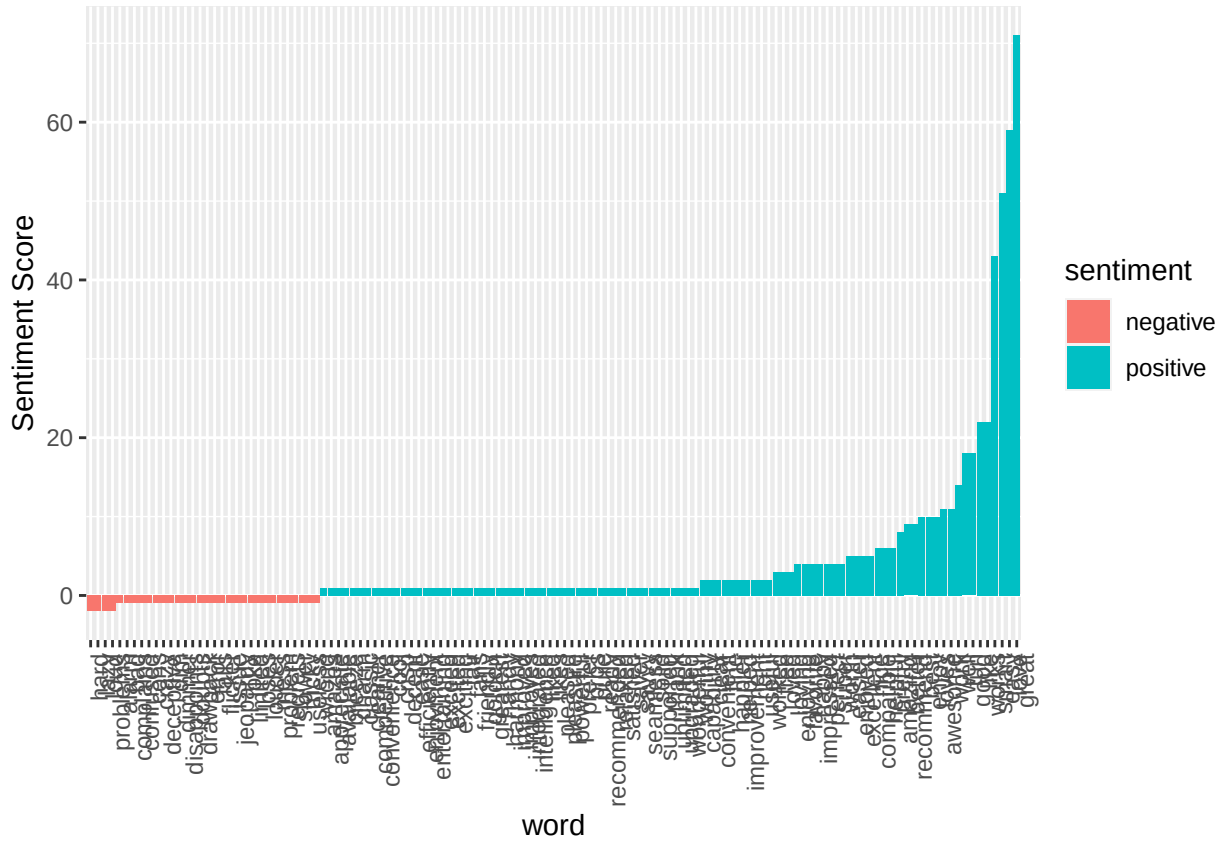
We will use `spread()` function to segregate the data into positive and negative sentiments. The `mutate()` function is then used to calculate the total sentiment (difference between positive and negative sentiment.)

A sample of the most common words :

```
## # A tibble: 6 x 3
##   word  sentiment     n
##   <chr> <chr>     <int>
## 1 great positive     71
## 2 love  positive     59
## 3 easy  positive     51
## 4 smart positive     43
## 5 like  positive     22
## 6 works positive     22
```

We can visualize the sentiment score :

```
counting_words %>%
  mutate(n = ifelse(sentiment == "negative", -n, n)) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n, fill = sentiment)) +
  geom_col() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(y = "Sentiment Score")
```



Alternately, we can also use a word cloud. For generating word clouds, we use the `wordcloud` package. In this cloud, positive words are green in color and negative words are red in color.

```
text_df %>%
  inner_join(bing) %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(colors = c("red", "dark green"),
                   max.words = 100)
```

Let us remove objects that we don't need.

```
rm(new_dat, dat, para, para1, text)
```

Now, to check if this product can be recommended to a new buyer, we can calculate the proportion of positive words to negative words.

```
## [1] "The proportion of positive sentiment to negative sentiment is :  3"
```

Let us check the actual proportion of positive reviews to negative reviews. To do this we can use the `reviews.doRecommend` object in the file.

```
## [1] "The actual proportion of positive to negative sentiment is :  41.14"
```

This model doesn't seem to be very accurate. This can be attributed to the fact that we are heavily reliant on the words having a place in the dictionary.

Model 2

Naive Bayes Approach

Naive Bayes is a Supervised Machine Learning algorithm based on the Bayes Theorem that is used to solve classification problems by following a probabilistic approach. It is based on the idea that the predictor variables in a Machine Learning model are independent of each other. Meaning that the outcome of a model depends on a set of independent variables that have nothing to do with each other. The idea is to take texts that are already classified as "positive" and "negative" and use Bayes' theorem to find the probability of words of unknown texts being positive/negative and sum over the probabilities for both categories to see which one "wins". Put differently, we find how probable positive/negative words are.

The mathematics behind Bayes Theorem

In general, Conditional probability is defined as the **likelihood of an event or outcome occurring**, based on the occurrence of a previous event or outcome. Conditional probability is calculated by multiplying the probability of the preceding event by the updated probability of the succeeding, or conditional, event.

The equation below represents the conditional probability of A, given B :

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Similarly the conditional probability of B, given A is :

$$P(B|A) = \frac{P(B \cap A)}{P(A)}$$

Therefore, on combining the two equations we get Bayes Theorem as,

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Bayes Theorem for Naive Bayes Algorithm

The above equation was for a single predictor variable, however, in real-world applications, there are more than one predictor variables and for a classification problem, there is more than one output class.

The classes can be represented as, $C_1, C_2, \ldots, C_k$ and the predictor variables can be represented as a vector, $x_1, x_2, \ldots, x_n$.

The objective of a Naive Bayes algorithm is to measure the conditional probability of an event with a feature vector $x_1, x_2, \ldots, x_n$ belonging to a particular class C_i.

$$P(C_i|x_1, x_2, \ldots, x_n) = \frac{P(x_1, x_2, \ldots, x_n)|C_i \cdot P(C_i)}{P(x_1, x_2, \ldots, x_n)} for 1 < i < k$$

On computing the above equation, we get,

$$P(x_1, x_2, \ldots, x_n|C_i) \cdot P(C_i) = P(x_1, x_2, \ldots, x_n, C_i)$$

$$P(x_1, x_2, \ldots, x_n, C_i) = P(x_1|x_2, \ldots, x_n, C_i) \cdot P(x_2, \ldots, x_n, C_i)$$

$$= P(x_1|x_2, \ldots, x_n, C_i) \cdot P(x_2|x_3, \ldots, x_n, C_i)P(x_3, \ldots, x_n, C_i) = \cdots =$$

$$P(x_1|x_2, \ldots, x_n, C_i) \cdot P(x_2|x_3, \ldots, x_n, C_i) \ldots (x_{n-1}|C_i) \cdot P(C_i)$$

However, the conditional probability, i.e., $P(x_j|x_{j+1}, \ldots, x_n, C_i)$ sums down to $P(x_j|C_i)$ since each predictor variable is independent in Naive Bayes.

The final equation comes down to :

$$P(C_i|x_1, x_2, \ldots, x_n) = (\prod_{j=1}^{j=n} P(x_j|C_i)) \cdot \frac{P(C_i)}{P(x_1, x_2, \ldots, x_n)} for 1 < i < k$$

Here, $P(x_1, x_2, \ldots, x_n)$ is constant for all the classes, we get,

$$P(C_i|x_1, x_2, \ldots, x_n) \alpha (\prod_{j=1}^{j=n} P(x_j|C_i)) \cdot P(C_i) for 1 < i < k$$

Implementing Naive Bayes Approach :

We use the `data_scored.csv` file. To implement Naive Bayes Approach, we will make use of the `tm`, `e1071`, `gmodels`, `SnowballC` and `crosstable` packages, among others.

For this, we use the data_scored.csv file. Each review is assigned a score of either 0 or 1 depending on whether is it negative or positive. It has two columns - sentence (the review) and score (its score).

```
data2_raw <- read.csv("C:/Users/MAG/Documents/CYO/data_scr.csv",
                      header = FALSE, col.names = c("sentence","score"),
                      stringsAsFactors = FALSE)
```

Overview of the data frame :

```
## 'data.frame':    591 obs. of  2 variables:
##  $ sentence: chr  "sentence" "Works great. Turns H&A up and down. Tv on and off. Lights on and off. A
##  $ score   : chr  "score" "1" "1" "1" ...
```

As we can see, the scores belong to the class `character`. We will convert them into factors.

```
data2_raw$score <- factor(data2_raw$score)
str(data2_raw$score)
```

```
##  Factor w/ 3 levels "0","1","score": 3 2 2 2 2 2 2 2 2 2 ...
```

We can see the total number of 1s and 0s :

```
##
##     0     1 score
##    13   577     1
```

We can see that only 2% of the reviews are negative.

Transforming the data :

We must now transform the data to make sure Naive Bayes functions as expected. To do this, we must create a **volatile corpus.** A corpus basically represents a collection of text documents. So a volatile corpus is a corpus that is stored in memory and which would be destroyed when the R object containing it is destroyed.

To create the volatile corpus we use the `tm` package.

```
data2_corpus <- VCorpus(VectorSource(data2_raw$sentence))
```

Let us look at the first message in the corpus :

```
## [1] "Works great. Turns H&A up and down. Tv on and off. Lights on and off. All from Alexa. You do hav
```

Now, we will pre-process this data, since it is not ready to have a model fitted onto yet.

First we will convert all words to lower case, since they may differ in format.

```
data2_corpus_clean <- tm_map(data2_corpus, content_transformer(tolower))
```

Let's see how to first sentence has changed :

```
## [1] "works great. turns h&a up and down. tv on and off. lights on and off. all from alexa. you do ha
```

Next we will remove any numbers :

```
# remove numbers
data2_corpus_clean <- tm_map(data2_corpus_clean, removeNumbers)
```

Stopwords are words in the data that appear often but serve no purpose. Some examples are 'to', 'and', 'from', 'I', "am', etc. In our data we have some additional stopwords such as 'Alexa', 'Echo' and 'Dot' since they are specific to the product. These are unnecessary and can be removed. This is done like this :

```
data2_corpus_clean <- tm_map(data2_corpus_clean, removeWords,
                             c(stopwords("english"),"Alexa","Echo","Dot"))
```

We then remove punctuation.

```
# remove punctuation
data2_corpus_clean <- tm_map(data2_corpus_clean, removePunctuation)
```

Stemming is another important process. In this, we remove the suffix and transform them into their root word. We will use the package `SnowballC` along with `tm`.

After that we remove whitespaces.

```
# stemming operation
data2_corpus_clean <- tm_map(data2_corpus_clean, stemDocument)

# remove whitespaces
data2_corpus_clean <- tm_map(data2_corpus_clean, stripWhitespace)
```

The first sentence now looks like this :

```
## [1] "work great turn h tv light alexa app special light"
```

Next we perform Tokenization using the `DocumentTermMatrix` function. A `DocumentTermMatrix` is a mathematical matrix that describes the frequency of terms that occur in a collection of documents. This is a matrix where. each row represents one document. each column represents one term (word)

```
data2_dtm <- DocumentTermMatrix(data2_corpus_clean)
```

We will split the data into train (80%) and test (20%) sets. The test set will be used to evaluate the algorithm. Excluding the column names (sentence and score) we have 590 observations.

80% of 590 is 472 and 20 of 590 is 118. Hence the first 472 observations form the training set and the last 118 observations form the test set.

```
data2_dtm_train <- data2_dtm[2:472, ]
data2_dtm_test <- data2_dtm[473:591, ]
```

Next we save the vectors that label the rows, in both sets

```
data2_train_labels <- data2_raw[2:472, ]$score
data2_test_labels <- data2_raw[473:591,]$score
```

The proportion of 0s in the train and test sets must be roughly equal. Let us check this.

```
print(paste("Proportion of 0s in train-set : ",
            prop.table(table(data2_train_labels))[1]))
```

```
## [1] "Proportion of 0s in train-set :  0.0212314225053079"
```

```
print(paste("Proportion of 0s in test-set : ",
            prop.table(table(data2_test_labels))[1]))
```

```
## [1] "Proportion of 0s in test-set :  0.0252100840336134"
```

They are nearly equal. That is, both the train and test sets have roughly 2% of negative reviews.

Generating a word cloud where frequently occurring words are colored blue and less frequent words are colored green :

```
wordcloud(data2_corpus_clean, max.words = 290, random.order = TRUE,
          colors = c("blue", "dark green"))
```

We can remove words from the matrix that appear less than 5 times and limit our `DocumentTermMatrix` to only include words that appear more than 5 times. Also, since Naive Bayes works with categorical data, we will convert the matrix to "yes" and "no" categorical variables.

```
data2_freq_words <- findFreqTerms(data2_dtm_train, 5)
str(data2_freq_words)
```

```
##  chr [1:254] ",äôm" ",äôs" ",äôve" "abil" "abl" "absolut" "account" "add" ...
```

```
data2_dtm_freq_train <- data2_dtm_train[ , data2_freq_words]
data2_dtm_freq_test <- data2_dtm_test[ , data2_freq_words]

convert_counts <- function(x) {
  x <- ifelse(x > 0, "Yes", "No")
}
```

Values that are greater than zero are assigned 'yes' and those *not above zero* are assigned 'no'.

```
data2_train <- apply(data2_dtm_freq_train, MARGIN = 2, convert_counts)
data2_test <- apply(data2_dtm_freq_test, MARGIN = 2, convert_counts)
```

We will now train the model using the `e1071` package :

```
data2_classifier <- naiveBayes(data2_train, data2_train_labels)
```

Predict and evaluate :

(CrossTable is a function that computes the descriptive statistics on datasets.)

```
data2_test_pred <- predict(data2_classifier, data2_test)
CrossTable(data2_test_pred, data2_test_labels,
           prop.chisq = FALSE, prop.t = FALSE, dnn = c('predicted', 'actual'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |-------------------------|
##
##
## Total Observations in Table:  119
##
##
##                  | data2_test_labels
## data2_test_pred |         0 |         1 | Row Total |
## ----------------|-----------|-----------|-----------|
##               1 |         3 |       116 |       119 |
## ----------------|-----------|-----------|-----------|
##    Column Total |         3 |       116 |       119 |
## ----------------|-----------|-----------|-----------|
##
##
```

Because of the nature of the data chosen it so happens that `data2_test_pred` set contains only 1s.

Let's calculate the accuracy of our prediction :

```
## [1] "Accuracy of the prediction is :  97.48"
```

The accuracy achieved in this model is very good. This means that the model has correctly classified all the reviews and has concluded that almost of them are positive. Hence we can recommend this item to a customer.

4 Results

Model Name	Result Description
Lexicon Based Approach	The model predicted that the ratio of the number of positive to negative reviews was only 3, when actually it was nearly 42. This would lead to uncertainty when trying to recommend the product to a new buyer. This gives the impression that, perhaps, 60% of the customers were dissatisfied. This would prevent new customers from buying it.
Naive Bayes Approach	This model gave very good results. The accuracy of its prediction was 97.48. This is definitely consistent with the 98% of customers who were happy with their purchase.

5 Conclusion

It is clear that the Naive Bayes Approach performed better than the Lexicon Based Approach. Choosing a different dataset would lead to different results. A possible downside in choosing this dataset is that the proportion of positive reviews is much larger than the proportion of negative reviews.

Advantages of the Naive Bayes Approach :

- This algorithm works quickly and can save a lot of time.
- Naive Bayes is suitable for solving multi-class prediction problems.
- If its assumption of the independence of features holds true, it can perform better than other models and requires much less training data.
- Naive Bayes is better suited for categorical input variables than numerical variables.

Disadvantages of the Naive Bayes Approach :

- Naive Bayes assumes that all predictors (or features) are independent, rarely happening in real life. This limits the applicability of this algorithm in real-world use cases.
- This algorithm faces the 'zero-frequency problem' where it assigns zero probability to a categorical variable whose category in the test data set wasn't available in the training dataset (although this could be overcome by using a smoothing technique).
- It cannot be applied to numerical variables.

Even so, it makes for a good model when used in sentiment analysis due to its high rate of success, high speed and efficiency.

6 References

[1] https://www.kaggle.com/datafiniti/consumer-reviews-of-amazon-products?select=Datafiniti_Amazon_Consumer_Reviews_of_Amazon_Products.csv

[2] https://towardsdatascience.com/a-hitchhikers-guide-to-sentiment-analysis-using-naive-bayes-classifier-b921c0fb694

[3] https://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences

[4] https://www.upgrad.com/blog/naive-bayes-explained/

[5] https://machinelearningmastery.com/types-of-classification-in-machine-learning/

[6] https://www.sciencedirect.com/science/article/pii/S2090447914000550