R Project – Sentiment Analysis

The aim of this project is to build a sentiment analysis model which will allow us to categorize words based on their sentiments, that is whether they are positive, negative and also the magnitude of it. Before we start with our R project, let us understand sentiment analysis in detail.

What is Sentiment Analysis?

Sentiment Analysis is a process of extracting opinions that have different polarities. By polarities, we mean positive, negative or neutral. It is also known as opinion mining and polarity detection. With the help of sentiment analysis, you can find out the nature of opinion that is reflected in documents, websites, social media feed, etc. Sentiment Analysis is a type of classification where the data is classified into different classes. These classes can be binary in nature (positive or negative) or, they can have multiple classes (happy, sad, angry, etc.).

Developing our Sentiment Analysis Model in R

We will carry out sentiment analysis with R in this project. The dataset that we will use will be provided by the R package ‘janeaustenR’.

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 of ‘sentiments’.

**require**("tidytext")

**library**(tidytext)

sentiments

## # A tibble: 6,786 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

## 7 abomination negative

## 8 abort negative

## 9 aborted negative

## 10 aborts negative

## # ... with 6,776 more rows

We will make use of three general purpose lexicons like –

AFINN bing loughran These three lexicons make use of the unigrams. Unigrams are a type of n-gram model that consists of a sequence of 1 item, that is, a word collected from a given textual data. In the AFINN lexicon model scores the words in a range from -5 to 5. The increase in negativity corresponds the negative sentiment whereas an increase in positivity corresponds the positive one. The bing lexicon model on the other hand, classifies the sentiment into a binary category of negative or positive. And finally, the loughran model that performs analysis of the shareholder’s reports. In this project, we will make use of the bing lexicons to extract the sentiments out of our data. We can retrieve these lexicons using the get\_sentiments() function. We can implement this as follows –

Performing Sentiment Analysis with the Inner Join

In this step, we will import our libraries ‘janeaustenr’, ‘stringr’ as well as ‘tidytext’. The janeaustenr package will provide us with the textual data in the form of books authored by the novelist Jane Austen. Tidytext will allow us to perform efficient text analysis on our data. We will convert the text of our books into a tidy format using unnest\_tokens() function.

We have performed the tidy operation on our text such that each row contains a single word. We will now make use of the “bing” lexicon to and implement filter() over the words that correspond to joy. We will use the book Sense and Sensibility and derive its words to implement out sentiment analysis model.

positive\_senti <- bing %>%

filter(sentiment == "positive")

tidy\_data %>%

filter(book == "Emma") %>%

semi\_join(positive\_senti) %>%

count(word, sort = TRUE)

## Joining, by = "word"

## # A tibble: 668 x 2

## word n

## <chr> <int>

## 1 well 401

## 2 good 359

## 3 great 264

## 4 like 200

## 5 better 173

## 6 enough 129

## 7 happy 125

## 8 love 117

## 9 pleasure 115

## 10 right 92

## # ... with 658 more rows

From our above result, we observe many positive words like “good”, “happy”, “love” etc. In the next step, we will use spread() function to segregate our data into separate columns of positive and negative sentiments. We will then use the mutate() function to calculate the total sentiment, that is, the difference between positive and negative sentiment.

**require**("tidyr")

**library**(tidyr)

Emma\_sentiment <- tidy\_data %>%

inner\_join(bing) %>%

count(book = "Emma" , index = linenumber %/% 80, sentiment) %>%

spread(sentiment, n, fill = 0) %>%

mutate(sentiment = positive - negative)

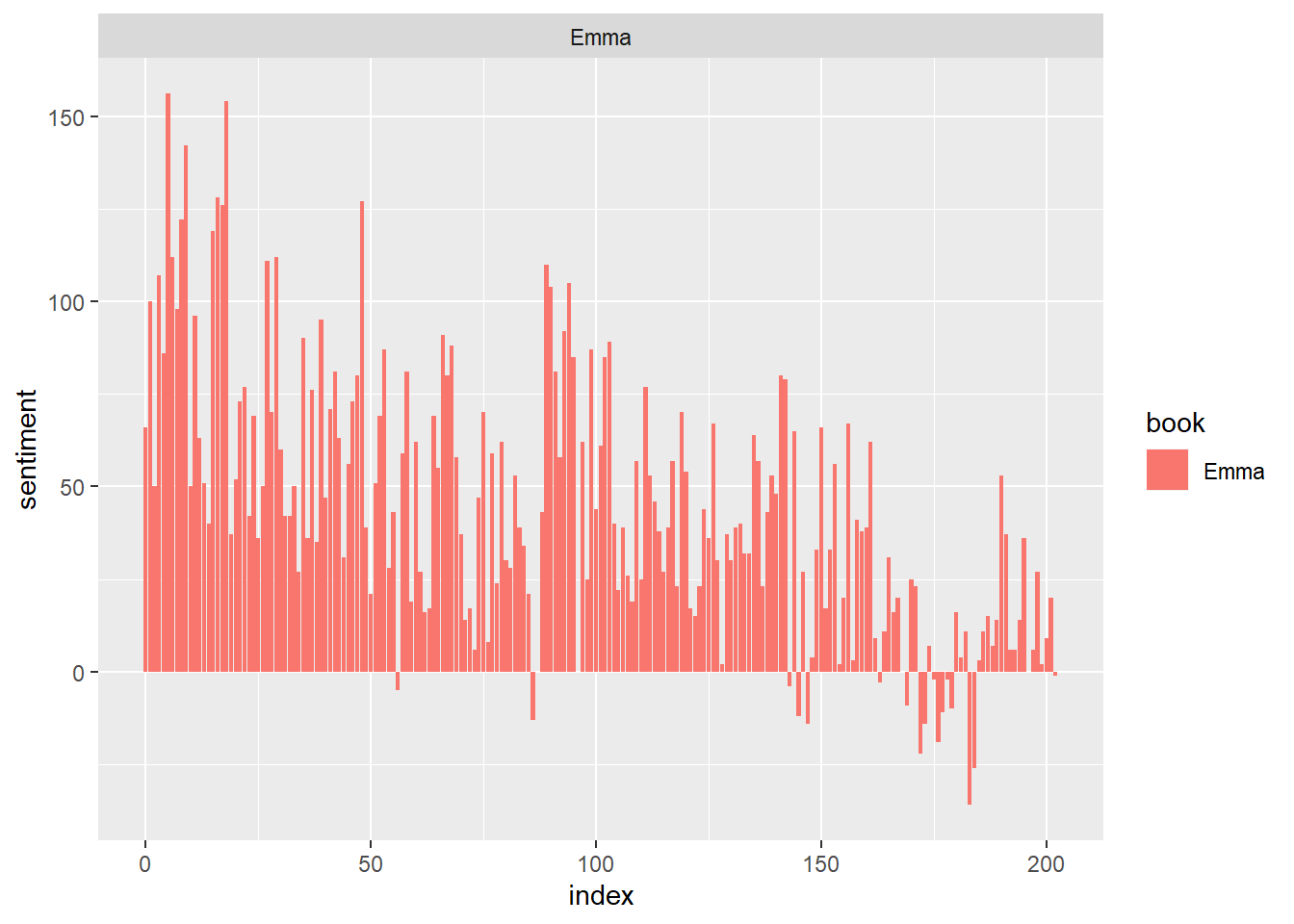
In the next step, we will visualize the words present in the book “Emma” based on their corrosponding positive and negative scores.

**library**(ggplot2)

ggplot(Emma\_sentiment, aes(index, sentiment, fill = book)) +

geom\_bar(stat = "identity", show.legend = TRUE) +

facet\_wrap(~book, ncol = 2, scales = "free\_x")



Let us now proceed towards counting the most common positive and negative words that are present in the novel.

counting\_words <- tidy\_data %>%

inner\_join(bing) %>%

count(word, sentiment, sort = TRUE)

## Joining, by = "word"

head(counting\_words)

## # A tibble: 6 x 3

## word sentiment n

## <chr> <chr> <int>

## 1 miss negative 1855

## 2 well positive 1523

## 3 good positive 1380

## 4 great positive 981

## 5 like positive 725

## 6 better positive 639

In the next step, we will perform visualization of our sentiment score. We will plot the scores along the axis that is labeled with both positive as well as negative words. We will use ggplot() function to visualize our data based on their scores.

counting\_words %>%

filter(n > 150) %>%

mutate(n = ifelse(sentiment == "negative", -n, n)) %>%

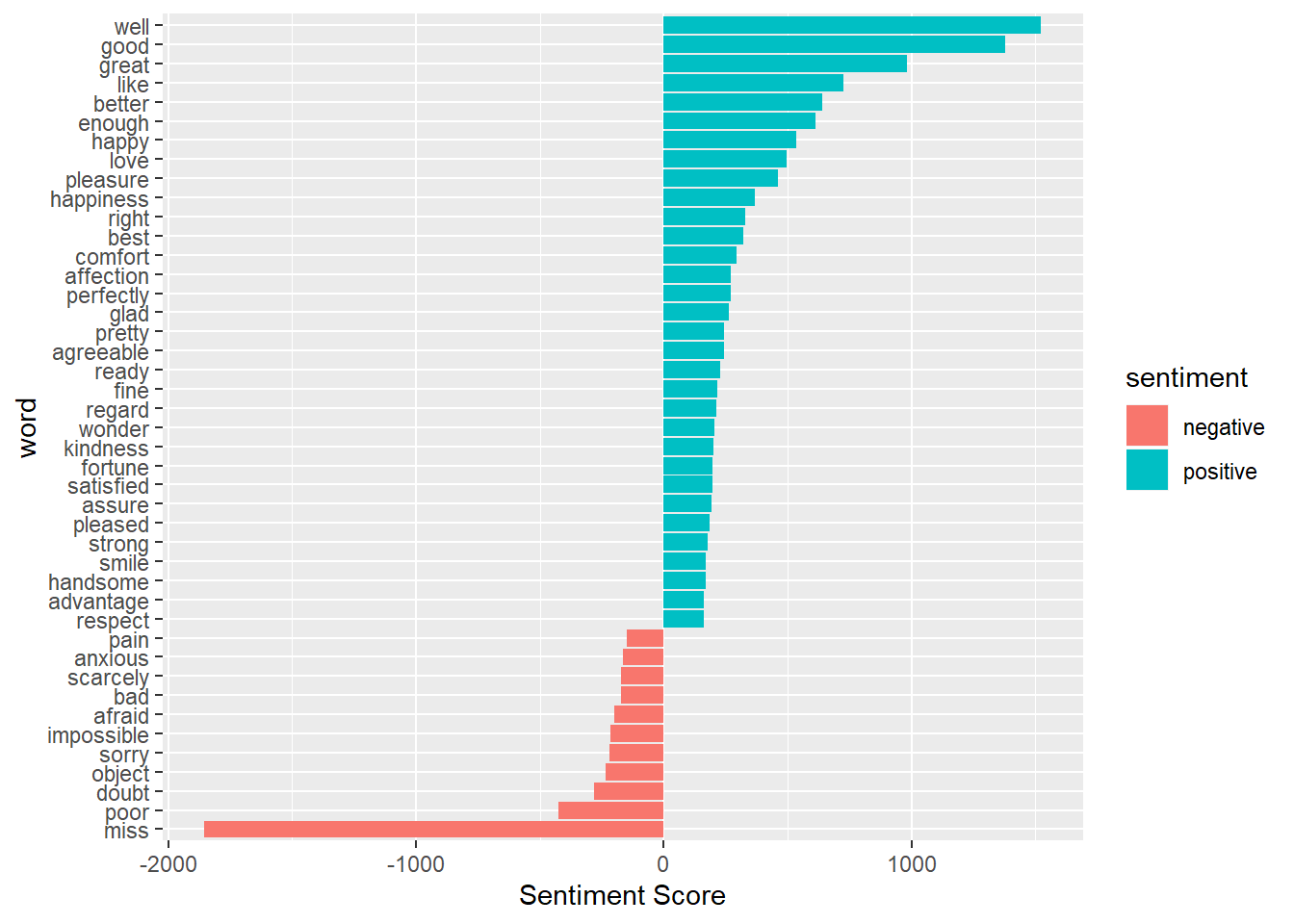
mutate(word = reorder(word, n)) %>%

ggplot(aes(word, n, fill = sentiment))+

geom\_col() +

coord\_flip() +

labs(y = "Sentiment Score")



In the final visualization, let us create a wordcloud that will delineate the most recurring positive and negative words. In particular, we will use the comparision.cloud() function to plot both negative and positive words in a single wordcloud as follows:

**require**("reshape2")

**require**("wordcloud")

**library**(reshape2)

**library**(wordcloud)

tidy\_data %>%

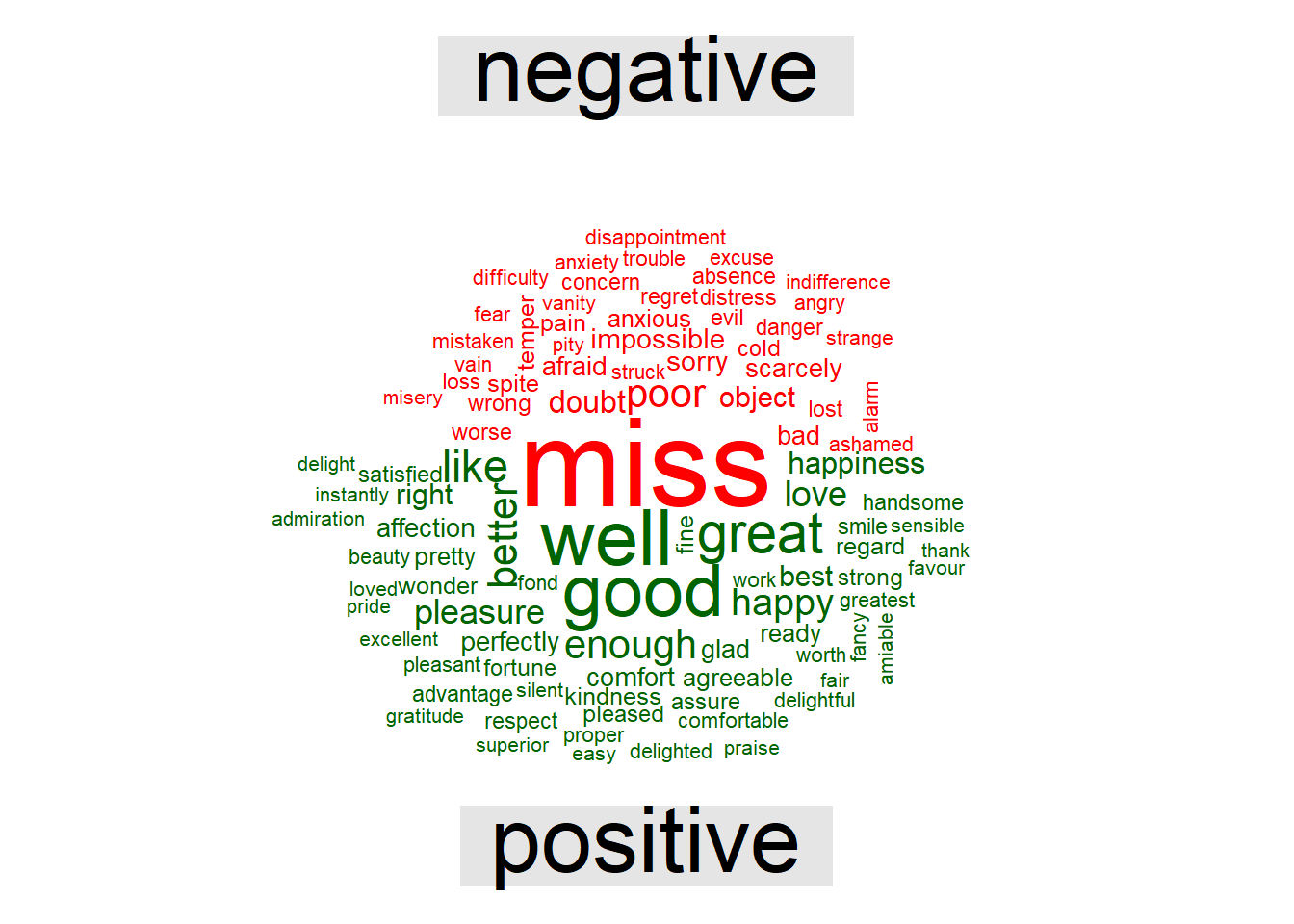
inner\_join(bing) %>%

count(word, sentiment, sort = TRUE) %>%

acast(word ~ sentiment, value.var = "n", fill = 0) %>%

comparison.cloud(colors = c("red", "dark green"),

max.words = 100)



*#load libraries*

**if** (!**require**("pacman")) install.packages("pacman")

pacman::p\_load(twitteR, wordcloud, tm, tidyr, tidytext, syuzhet, ngram, NLP, RColorBrewer, RTextTools, e1071, caret, knitr)

*#read data*

sentiment = read.csv("movie\_review\_sent.csv", stringsAsFactors = F)

Data Cleaning

The data will have a bunch of stuff not conveying any kind of sentiments like names, punctuation and others. It is better to remove them before moving forward. some custom cleaning might be required but the code will take care most common data cleaning steps.

*### clean data ####*

text = sentiment$text *# save tweets to another data set "text"*

text = gsub("(RT|via)((?:\\b\\W\*@\\w+)+)","",text) *#remove names*

text = gsub("http[^[:blank:]]+","",text) *#remove html links*

text = gsub("@\\w+","",text) *#remove people names*

text = gsub("[[:punct:]]","",text) *#remove punctuations*

text = trimws(text, which = c("both", "left", "right")) *# remove whitespace*

text = gsub('[[:digit:]]+', '', text) *# remove digits*

text = gsub("[\r\n]", "", text) *# remove line breaks*

text = iconv(text, to = "ASCII//TRANSLIT") *# remove not readable standard text*

text = iconv(text, "ASCII", "UTF-8", sub="") *# remove not readable standard text*

text = tolower(text) *# lower case*

sentiment$text = text

Exploratory Analysis

Word cloud is quick and effective method to visualize word frequency and identify most repeated words in the corpus. One must, however, be vigilant while making word clouds. If not used properly they often paint a misleading picture.

*# dependent variable*

kable(table(sentiment$Positive), col.names = c("Positive", "Frequency"), align = "l")

| **Positive** | **Frequency** |
| --- | --- |
| 0 | 586 |
| 1 | 983 |

*#independent variable - word cloud*

corpus = Corpus(VectorSource(text)) *# convert tweets to corpus*

*#some more cleaning*

corpus = tm\_map(corpus, removeWords, stopwords("english")) *#remove stopwords like "and","the" and"that"*

corpus = tm\_map(corpus, stripWhitespace) *# remove whitespace*

*# word frequency*

uniqwords = as.matrix(TermDocumentMatrix(corpus)) *# covert corpus to Term Document matrix*

wordfreq = sort(rowSums(uniqwords),decreasing=TRUE) *# Count frequency of words in each tweet*

WCinput = data.frame(word = names(wordfreq),freq=wordfreq) *# word frequency dataframe*

*#generate the wordcloud*

wordcloud(words = WCinput$word, freq = WCinput$freq, min.freq = 2,

max.words=200, random.order=FALSE, rot.per=0.35,

colors=brewer.pal(8, "Dark2"))



Data Preparation

Before we move on to training the classifier, we need do split the data into training and validation sets. We will create a document matrix with all the unique words with a frequency of more than 5 in the reviews and use them as independent variables in the our classifier. We also need to ensure that data is in correct format.

Data Processing

*#covert dependent to factor type*

sentiment$Positive = as.factor(sentiment$Positive)

*#create corpus*

corpus = Corpus(VectorSource(sentiment$text))

*#create doucment*

NBinput = DocumentTermMatrix(corpus)

*# Partion the data into trainin and validation*

Index = sample(1:nrow(sentiment), size = round(0.7\*nrow(sentiment)), replace=FALSE)

text.train = sentiment[Index ,]

text.test = sentiment[-Index ,]

*# doc*

doc.train = NBinput[Index,] *#document matrix for training*

doc.test = NBinput[-Index,] *#document matrix for test*

*#corpus*

corpus.train = corpus[Index] *#corpus for training*

corpus.test = corpus[-Index] *# corpus for test*

Limit Independent Varaibles in the model

*## There are many unique words, lets only use those words which occur more than 5 times in the document for NB*

fivefreq = findFreqTerms(doc.train, 5) *# generate list of words which occur 5 times or more*

*# restrict the document to have only those words which occur five times or more*

dOC.train.nb = DocumentTermMatrix(corpus.train, control=list(dictionary = fivefreq))

doc.test.nb = DocumentTermMatrix(corpus.test, control=list(dictionary = fivefreq))

*#Convert frequency count to "No" or "Yes"*

convert\_count = **function**(x) {

y = ifelse(x > 0, 1,0)

y = factor(y, levels=c(0,1), labels=c("No", "Yes"))

y

}

*# create training and validation features*

trainNB = apply(dOC.train.nb, 2, convert\_count)

testNB = apply(doc.test.nb, 2, convert\_count)

Train NB Classifier

Now that we have our data as we need, training the model becomes very simple. We just need to use naiveBayes function for this purpose with trainNB data set.

*#run naive bayes*

classifier = naiveBayes(trainNB, factor(text.train$Positive), laplace = 1)

Prediction and Confusion Matrix

To understand how good the classifier is, we will predict sentiments in test data set. A confusion matrix will then give us metric like overall accuracy, sensitivity and specificity. The model performance according to these metrics are satisfactory.

*#predict using Naive Bayes*

pred = predict(classifier, newdata=testNB)

*#confusion matrix*

confusionMatrix(pred, text.test$Positive)

## Confusion Matrix and Statistics

##

## Reference

## Prediction 0 1

## 0 135 126

## 1 39 171

##

## Accuracy : 0.6497

## 95% CI : (0.6047, 0.6928)

## No Information Rate : 0.6306

## P-Value [Acc > NIR] : 0.209

##

## Kappa : 0.3186

## Mcnemar's Test P-Value : 2.155e-11

##

## Sensitivity : 0.7759

## Specificity : 0.5758

## Pos Pred Value : 0.5172

## Neg Pred Value : 0.8143

## Prevalence : 0.3694

## Detection Rate : 0.2866

## Detection Prevalence : 0.5541

## Balanced Accuracy : 0.6758

##

## 'Positive' Class : 0