Sentiment Analysis for Amazon Review and Drug Review

fa22-prj-rongl2-xdai12-zixingd2

2022-12-07

Contents

1 Introduction

In recent years, machine learning techniques have become increasingly popular and relevant to solving text and sentiment-related problems. It has boosted performance on several tasks and significantly reduced the necessity for human efforts. For this project, we focused on text classification, especially sentiment analysis, on two datasets, *Amazon Review* and *Drug Review*. Although the *Amazon Review* dataset is popular and was being used for many research papers and projects, most code works were carried out using Python. Therefore, we decided to explore more on implementing four classic Natural Language Processing (NLP) methods using R packages. We first replicated the R code from existing literature for the *Amazon Review* dataset. We adapted them to a newer but less popular UCI Machine Learning Repository dataset. Our goal for the project is to compare classifiers, including `BoW`, `Word2Vec`, `GloVe`, `fastText` for two different datasets.

2 Data

2.1 Dataset Overview

For the *Amazon Review* dataset, we use the dataset constructed and made available by Zhang, Zhao, and LeCun (2015). The dataset contains about 1,800,000 training samples and 200,000 testing samples with three attributes, which are classification labels (1 for negative reviews and 2 for positive reviews), the title of each review text, and the review text body. Due to the limit of computer computation ability, we pulled out the first 100,000 data samples and split 80% of the data into the training set and 20% of the data into the testing set.

For the Drug dataset, we downloaded from the UCI Machine Learning Repository. The dataset has 215,063 samples with 6 attributes, including drugName, condition, review, rating (1 to 10), date, usefulCount. Similar to *Amazon Review* dataset, we split the whole dataset into training (80%) and testing (20%) datasets. In order to replicate the code, we categorized the dataset into two labels, where rating range from [1 to 4] are categorized as [1(negative)] and the rating range from [7 to 10] are categorized as [2(positive)]. We combined the text columns together and removed symbols that are not a number or a word. Moreover, we only retained the columns of rating and merged columns of review, name, condition as one text body attribute. Hence our drug dataset has the same format as *Amazon Review*.

2.2 Dataset Prepossessing

Take the code for Amazon reviews as example.

2.2.1 Amazon reviews

We first set seed and read the data. Note that we split the original data into training and testing according to ratio 8:2.

```
set.seed(1)
N_amazon <- 100000
N_train_amazon <- 0.8*N_amazon
reviews_amazon <- readLines("amazon_review_polarity_csv/train.csv", n = N_amazon)
reviews_amazon <- data.frame(reviews_amazon)
```

Then we separating the sentiment and the review text.

```r
library(tidyr)
reviews_amazon <- separate(data = reviews_amazon, col = reviews_amazon,
                           into = c("Sentiment", "SentimentText"), sep = 4)
```

Here we got the data before prepossessing.

	Sentiment	SentimentText
1	"2",	"Stuning even for the non-gamer","This sound track ...
2	"2",	"The best soundtrack ever to anything.","I'm reading a...
3	"2",	"Amazing!","This soundtrack is my favorite music of a...
4	"2",	"Excellent Soundtrack","I truly like this soundtrack an...
5	"2",	"Remember, Pull Your Jaw Off The Floor After Hearing...

Figure 1: Head rows of the data before prepossessing

Since the unnecessary punctuation may cause the problem for our sentiment analysis, we remove the punctuation with the following code:

```r
# Retaining only alphanumeric values in the sentiment column
reviews_amazon$Sentiment <- gsub("[^[:alnum:] ]","",reviews_amazon$Sentiment)
# Retaining only alphanumeric values in the sentiment text
reviews_amazon$SentimentText <- gsub("[^[:alnum:] ]"," ",reviews_amazon$SentimentText)
# Replacing multiple spaces in the text with single space
reviews_amazon$SentimentText <- gsub("(?<=[\\s])\\s*|^\\s+|\\s+$", "",
                                 reviews_amazon$SentimentText, perl=TRUE)
# Writing the output to a file that can be consumed in other models
write.table(reviews_amazon,file = "Sentiment Analysis Dataset.csv",row.names = F,
            col.names = T,sep=',')
```

This will give us the following output:

	Sentiment	SentimentText
1	2	Stuning even for the non gamer This sound track was...
2	2	The best soundtrack ever to anything I m reading a lo...
3	2	Amazing This soundtrack is my favorite music of all ti...
4	2	Excellent Soundtrack I truly like this soundtrack and I ...
5	2	Remember Pull Your Jaw Off The Floor After Hearing i...

Figure 2: Head rows of the data after prepossessing

Since The fastText algorithm expects the dataset to be in a format: ___ label ___ <x> <text>, where x is the class name and text is the review text, we need to transform our data format as required.

```r
reviews_amazon <- readLines('amazon_review_polarity_csv/train.csv', n = N_amazon)
# Basic EDA to confirm that the data is read correctly
print(class(reviews_amazon))
```

```
## [1] "character"
```

```r
print(length(reviews_amazon))
```

```
## [1] 100000
```

```r
# Replacing the positive sentiment value 2 with __label__2
reviews_amazon <- gsub("\\\"2\\\",","__label__2 ",reviews_amazon)
# Replacing the negative sentiment value 1 with __label__1
reviews_amazon <- gsub("\\\"1\\\",","__label__1 ",reviews_amazon)
# Removing the unnecessary \" characters
reviews_amazon <- gsub("\\\""," ",reviews_amazon)
# Replacing multiple spaces in the text with single space
reviews_amazon <- gsub("(?<=[\\s])\\s*|^\\s+|\\s+$", "", reviews_amazon, perl=TRUE)

# Writing the revamped file to the directory so we could use it with
# fastText sentiment analyzer project
fileConn <- file("Sentiment Analysis Dataset_ft.txt")
writeLines(reviews_amazon, fileConn)
close(fileConn)
```

This will give us the following dataset:

	V1
1	__label__2 Stuning even for the non-gamer , This sou...
2	__label__2 The best soundtrack ever to anything. , I'm...
3	__label__2 Amazing! , This soundtrack is my favorite ...
4	__label__2 Excellent Soundtrack , I truly like this soun...
5	__label__2 Remember, Pull Your Jaw Off The Floor Aft...

Figure 3: Head rows of the prepossessing data for FastText

2.2.2 Drug Review

The prepossing method for Drug Review are similar to Amazon Review.

However we did find some problems during our training processes, which is the Drug Review are actually imbalanced. So we need to balanced our Drug Review, make them equal numbers of the positive reviews and negative reviews.

```r
# Checking the summary of our label for Drug data
(Sentimentable = table(reviews_text_Drug$Sentiment))
```

```
##
##      1      2
##  40075 106866
```

```r
# Balance our Drug data
minlabel <- names(which(Sentimentable == min(Sentimentable)))
maxlabel <- names(which(Sentimentable == max(Sentimentable)))

n_maxlabel <- min(Sentimentable)
minlabelid <- c(1:N_Drug)[reviews_text_Drug$Sentiment==minlabel]
maxlabelid <- sample(c(1:N_Drug)[reviews_text_Drug$Sentiment==maxlabel],n_maxlabel)
balanceid <- sample(c(minlabelid,maxlabelid))
reviews_text_Drug <- reviews_text_Drug[balanceid,]

N_Drug <- nrow(reviews_text_Drug)
N_train_Drug <- round(0.8*N_Drug)
```

3 BoW approach with naive bayes

Bag of Words (BoW) method is widely used in NLP and computer vision fields. It takes the occurrence of each word in the text regardless of grammar and makes it into "bags" to characterize the text. To implement BoW method for our dataset, *Amazon Review* and *Drug Review*, we first use `VCorpus` function and `DocumentTermMatrix` function in the `tm` package to convert text into a Document Term Matrix (DTM). By adjusting the built-in parameter in the `DocumentTermMatrix` function, we do not have to worry about cleaning the dataset with stop words. In order to make the model more precise, we removed words that do not occur in 99% of the documents by using `removeSparseTerms` function.

After finishing the process of BoW conversion, we can use the DTM to create word clouds for both positive and negative sentiment cases. And also to better interpret our word cloud, we simply use the two-sample t-test to better discriminant our words. Finally, we followed Chinnamgari (2019) and used the Naive Bayes sentiment classifier to perform predictions. Utilizing the `nb_sent_classifier` in the `e1071` package, we obtained the prediction results with approximate **81.19%** for *Amazon Review* data and **74.77%** for *Drug Review*.

3.1 Amazon reviews

```r
library(SnowballC)
library(tm)
# Reading the transformed file as a dataframe
text_amazon <- read.table(file='Sentiment Analysis Dataset.csv', sep=',', header = TRUE)

# Transforming the text into volatile corpus
amazon_corp <- VCorpus(VectorSource(text_amazon$SentimentText))
```

```r
# Creating document term matrix (DTM)
dtm_amazon <- DocumentTermMatrix(amazon_corp, control = list( tolower = TRUE,
   removeNumbers = TRUE, stopwords = TRUE, removePunctuation = TRUE, stemming = TRUE))
# Basic EDA on dtm
inspect(dtm_amazon)
```

```
## <<DocumentTermMatrix (documents: 100000, terms: 74760)>>
## Non-/sparse entries: 3399444/7472600556
## Sparsity           : 100%
```

```
## Maximal term length: 188
## Weighting          : term frequency (tf)
## Sample             :
##        Terms
## Docs    book get good great just like movi one read time
##   1250     0   0    0     0    0    0    0   0    0    0
##   56817    0   0    0     0    0    0    0   0    0    0
##   63995    0   2    1     1    0    2    0   2    0    1
##   6785     0   7    1     0    0    1    0   1    0    0
##   69262    0   0    0     0    0    0    0   0    0    0
##   73633    1   0    0     2    0    3    0   2    0    2
##   79144    0   0    0     0    0    0    0   0    0    0
##   80872    0   0    0     0    0    0    0   0    0    0
##   85894    0   1    0     0    0    1    0   1    0    0
##   87875    0   0    0     0    0    0    0   0    0    0
```

We see that the DTM is 100% sparse. Since the DTM tends to get very big, we removed sparse terms, that is, terms occurring only in very few documents, and tried to reduce the size of the matrix without losing significant relations inherent to the matrix.

```
# Removing sparse terms
dtm_amazon = removeSparseTerms(dtm_amazon, 0.99)
inspect(dtm_amazon)
```

```
## <<DocumentTermMatrix (documents: 100000, terms: 645)>>
## Non-/sparse entries: 2131029/62368971
## Sparsity           : 97%
## Maximal term length: 10
## Weighting          : term frequency (tf)
## Sample             :
##        Terms
## Docs    book get good great just like movi one read time
##   34297    0   1    0     0    2    5    0   3    0    0
##   38984    6   0    0     1    1    0    0   1    0    1
##   42051    3   0    1     0    2    1    0   3    5    1
##   56269    0   1    0     0    1    0    1   1    0    1
##   65117    0   1    1     1    1    0    0   1    0    0
##   65135    0   0    1     2    0    4    0   0    1    0
##   6785     0   7    1     0    0    1    0   1    0    0
##   80366    0   3    1     1    1    1    0   8    0    0
##   87149    6   0    1     0    0    1    0   2    1    0
##   90397    0   2    1     0    3    3    0   2    0    0
```

Using the DTM, we can create word clouds for better understanding for our sentiment text. However, we found that some of the selected words are difficult to interpret. So we decided to first filter out a vocab with words that we can interpret. We used a simple screening method, which we described in discriminant analysis before: two-sample t-test. Assume we have one-dimensional observations from two groups

$$X_1, X_2, \ldots, X_m, \quad Y_1, Y_2, \ldots, Y_n X$$

to test whether the X population and the Y population have the same mean, we compute the following two-sample t-statistic $\frac{\bar{X}-\bar{Y}}{\sqrt{\frac{s_X^2}{m}+\frac{s_Y^2}{n}}}$.

where s_X^2 denotes the sample variance of X.

Again, we use the training data from the first split. Since dtm_train is a large sparse matrix, I use commands from the R package slam to efficiently compute the mean and var for each column of dtm_train.

```
# Word Cloud preparing
v.size = dim(dtm_amazon)[2]
ytrain = as.numeric(text_amazon$Sentiment)
```

```
# Using two-sample t-test to find the most represent words to show our Word Cloud
library(slam)
summ = matrix(0, nrow=v.size, ncol=4)
summ[,1] = colapply_simple_triplet_matrix(
  as.simple_triplet_matrix(dtm_amazon[ytrain==2, ]), mean)
summ[,2] = colapply_simple_triplet_matrix(
  as.simple_triplet_matrix(dtm_amazon[ytrain==2, ]), var)
summ[,3] = colapply_simple_triplet_matrix(
  as.simple_triplet_matrix(dtm_amazon[ytrain==1, ]), mean)
summ[,4] = colapply_simple_triplet_matrix(
  as.simple_triplet_matrix(dtm_amazon[ytrain==1, ]), var)

n1 = sum((ytrain)-1);
n = length(ytrain)
n0 = n - n1

myp = (summ[,1] - summ[,3])/
  sqrt(summ[,2]/n1 + summ[,4]/n0)
```

We ordered words by the magnitude of their t-statistics, which are then divided into two lists: positive words and negative words.

```
words = colnames(dtm_amazon)
id = order(abs(myp), decreasing=TRUE)
pos.list = words[id[myp[id]>0]]
posvalue = myp[id][myp[id]>0][1:50]
neg.list = words[id[myp[id]<0]]
negvalue = myp[id][myp[id]<0][1:50]
```

Using the `wordcloud` package to plot the Word Cloud for most represent words, both positive and negative.

```
# Word Cloud for positive words
library(wordcloud)
wordcloud(words = pos.list[1:50], freq = posvalue, scale=c(6,.2), min.freq = 5,
          random.order=FALSE, rot.per=0.35, colors = brewer.pal(8, "Dark2"))
```

```
# Word Cloud for negative words
wordcloud(words = neg.list[1:50], freq = abs(negvalue), scale=c(4.3,.2), min.freq = 5,
          random.order=FALSE, rot.per=0.35, colors = brewer.pal(8, "Dark2"))
```

```
library(png)
par(mfrow=c(1, 2), mar=c(1, 0, 3, 0))
plot.new()
plot.window(xlim=c(0, 1), ylim=c(0, 1), asp=1)
```

```
rasterImage(readPNG("amazonpos"), 0, 0, 1, 1)
title('Positive words', line = -0.5)
plot.new()
plot.window(xlim=c(0, 1), ylim=c(0, 1), asp=1)
rasterImage(readPNG("amazonneg"), 0, 0, 1, 1)
title('Negative words', line = -0.5)
title("Word Clouds from Amazon reviews", line = -22, outer = TRUE)
```

Positive words **Negative words**

Word Clouds from Amazon reviews

Then, we continue using the DTM that to training with machine learning classification. We divide our DTM into training (80%) and testing (20%) datasets.

```
# Splitting the train and test DTM
dtm_amazon_train <- dtm_amazon[1:N_train_amazon, ]
dtm_amazon_test <- dtm_amazon[(N_train_amazon+1):N_amazon, ]
dtm_amazon_train_labels <- as.factor(as.character(text_amazon[1:N_train_amazon, ]$Sentiment))
dtm_amazon_test_labels <- as.factor(as.character(text_amazon[(N_train_amazon+1):N_amazon, ]$Sentiment))
```

Here we use Naive Bayes to create a classifier. Since Naive Bayes is generally trained on data with nominal features, DTM need to be converted to nominal prior to feeding the dataset as input for creating the model with Naive Bayes.

```
# Convert the cell values with a non-zero value to Y, and in case of a zero we convert it to N
cellconvert<- function(x) { x <- ifelse(x > 0, "Y", "N") }

# Applying the function to rows in training and test datasets
```

```r
dtm_amazon_train <- apply(dtm_amazon_train, MARGIN = 2,cellconvert)
dtm_amazon_test <- apply(dtm_amazon_test, MARGIN = 2,cellconvert)
```

Then, we proceed to build a text sentiment analysis classifier using the Naive Bayes algorithm from the `e1071` package.

```r
# Training the naive bayes classifier on the training dtm
library(e1071)
nb_amazon_senti_classifier <- naiveBayes(dtm_amazon_train,dtm_amazon_train_labels)
# Printing the summary of the model created
summary(nb_amazon_senti_classifier)
```

```
##           Length Class  Mode
## apriori     2    table  numeric
## tables    645    -none- list
## levels      2    -none- character
## isnumeric 645    -none- logical
## call        3    -none- call
```

Finally, we used the model object to predict sentiment on the test data DTM.

```r
# Making predictions on the test data dtm
nb_amazon_predicts <- predict(nb_amazon_senti_classifier, dtm_amazon_test, type="class")

# Computing accuracy of the model
library(rminer)
print(mmetric(nb_amazon_predicts, dtm_amazon_test_labels, c("ACC")))
```

```
## [1] 81.19
```

3.2 Drug Data

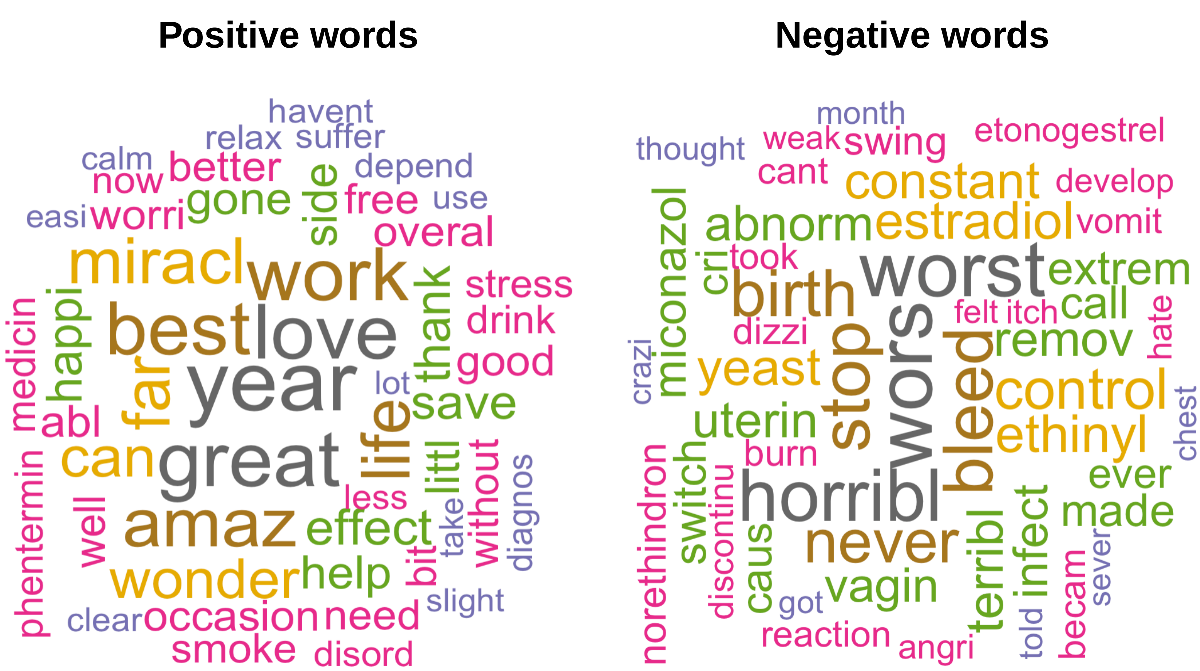
```
## [1] 80150     2
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 80150
```

```
## <<DocumentTermMatrix (documents: 80150, terms: 44610)>>
## Non-/sparse entries: 2919201/3572572299
## Sparsity           : 100%
## Maximal term length: 95
## Weighting          : term frequency (tf)
## Sample             :
##        Terms
## Docs    day effect get month pain start take week work year
##   14443   0      7   0     0    0     1    1    0    0    0
##   21739   3      4   3     1   10     4    3    5    6    1
##   32948   9      1   1     1    1     3    5    1    1    5
##   35157   7      7   4     1    3     1    6    4    4    0
```

```
##   39889  0       2   2     4    3     4    1    0    2    1
##   4810   0       7   0     0    0     1    1    0    0    0
##   48674  7       7   4     1    3     1    6    4    4    0
##   50714  7       5   3     2    0     4    9    6    2    1
##   56489  0       2   2     4    3     4    1    0    2    1
##   79862  6       0   2     0    7     0    2    0    4    2

## <<DocumentTermMatrix (documents: 80150, terms: 645)>>
## Non-/sparse entries: 2177799/49518951
## Sparsity           : 96%
## Maximal term length: 14
## Weighting          : term frequency (tf)
## Sample             :
##        Terms
## Docs    day effect get month pain start take week work year
##   1194  2       3   4     0    0     1    3    3    1    0
##   21739 3       4   3     1   10     4    3    5    6    1
##   32948 9       1   1     1    1     3    5    1    1    5
##   35157 7       7   4     1    3     1    6    4    4    0
##   35179 2       3   4     0    0     1    3    3    1    0
##   39889 0       2   2     4    3     4    1    0    2    1
##   48674 7       7   4     1    3     1    6    4    4    0
##   50714 7       5   3     2    0     4    9    6    2    1
##   56489 0       2   2     4    3     4    1    0    2    1
##   79862 6       0   2     0    7     0    2    0    4    2
```



Word Clouds from Drug reviews

```
##           Length Class  Mode
## apriori     2    table  numeric
## tables    645    -none- list
## levels      2    -none- character
## isnumeric 645    -none- logical
## call        3    -none- call
```

```
## [1] 74.7723
```

4 Pretrained word2vec word embedding with random forest algorithm

4.1 Amazon reviews

```
library(softmaxreg)
# Importing the word2vec pretrained vector into memory
data(word2vec)
dim(word2vec)
```

```
## [1] 12853    21
```

```
# Function to get word vector for each review
docVectors <- function(x) { wordEmbed(x, word2vec, meanVec = TRUE) }
text_amazon <- read.csv(file='Sentiment Analysis Dataset.csv', header = TRUE)
# Applying the docVector function on each of the reviews
# Storing the matrix of word vectors as temp
temp_amazon <- t(sapply(text_amazon$SentimentText, docVectors))
dim(temp_amazon)
```

```
## [1] 100000     20
```

```
# Splitting the dataset into train and test
temp_amazon_train <- temp_amazon[1:N_train_amazon,]
temp_amazon_test <- temp_amazon[(N_train_amazon+1):N_amazon,]
labels_amazon_train <- as.factor(as.character(text_amazon[1:N_train_amazon,]$Sentiment))
labels_amazon_test <- as.factor(as.character(text_amazon[(N_train_amazon+1):N_amazon,]$Sentiment))
library(randomForest)
# Training a model using random forest classifier with training dataset
# Observe that we are using 20 trees to create the model
rf_amazon_senti_classifier <- randomForest(temp_amazon_train, labels_amazon_train, ntree=20)
print(rf_amazon_senti_classifier)
```

```
##
## Call:
##  randomForest(x = temp_amazon_train, y = labels_amazon_train,      ntree = 20)
##                Type of random forest: classification
##                      Number of trees: 20
## No. of variables tried at each split: 4
```

```
##
##         OOB estimate of  error rate: 40%
## Confusion matrix:
##       1     2 class.error
## 1 23547 15436   0.3959675
## 2 16563 24448   0.4038673
```

```
# Making predictions on the dataset
rf_amazon_predicts <- predict(rf_amazon_senti_classifier, temp_amazon_test)
library(rminer)
print(mmetric(rf_amazon_predicts, labels_amazon_test, c("ACC")))
```

```
## [1] 62.555
```

4.2 Drug Data

```
## [1] 12853    21
```

```
## [1] 80150    20
```

```
##
## Call:
##  randomForest(x = temp_train_Drug, y = labels_train_Drug, ntree = 20)
##                Type of random forest: classification
##                      Number of trees: 20
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 31.81%
## Confusion matrix:
##       1     2 class.error
## 1 23040  8945   0.2796623
## 2 11453 20677   0.3564581
```

```
## [1] 70.98565
```

5 GloVe word embedding

Pennington, Socher, and Manning (2014) developed an extension of the `word2vec` method called `GloVe Vectors` for Word Representation (`GloVe`) for efficiently learning word vectors.

`GloVe` combines the global statistics of matrix factorization techniques, such as LSA, with the local context-based learning in `word2vec`. Also, unlike `word2vec`, rather than using a window to define local context, `GloVe` constructs an explicit word context or word co-occurrence matrix using statistics across the whole text corpus. As an effect, the learning model yields generally better word embeddings.

The `text2vec` package in R has a `GloVe` implementation that we could use to train to obtain word embeddings from our own training corpus. The following code block demonstrates how `GloVe` word embeddings can be created and used for sentiment analysis. ## Amazon reviews

```r
# Including the required library
library(text2vec)
# Reading the dataset
text_amazon <- read.csv(file='Sentiment Analysis Dataset.csv', header = TRUE)
# Subsetting only the review text so as to create Glove word embedding
wiki_amazon <- as.character(text_amazon$SentimentText)
# Create iterator over tokens
tokens_amazon <- space_tokenizer(wiki_amazon)
# Create vocabulary. Terms will be unigrams (simple words).
it_amazon <- itoken(tokens_amazon, progressbar = FALSE)
vocab_amazon <- create_vocabulary(it_amazon)
# Consider a term in the vocabulary if and only if the term has appeared at least
# three times in the dataset
vocab_amazon <- prune_vocabulary(vocab_amazon, term_count_min = 3L)
# Use the filtered vocabulary
vectorizer_amazon <- vocab_vectorizer(vocab_amazon)
# Use window of 5 for context words and create a term co-occurance matrix
tcm_amazon <- create_tcm(it_amazon, vectorizer_amazon, skip_grams_window = 5L)
# Create the glove embedding for each in the vocab and
# the dimension of the word embedding should set to 50
# x_max is the maximum number of co-occurrences to use in the weighting function
glove <- GlobalVectors$new(rank = 50, x_max = 100)
wv_main_amazon <- glove$fit_transform(tcm_amazon, n_iter = 10, convergence_tol = 0.01)
```

```
## INFO  [22:05:12.537] epoch 1, loss 0.0502
## INFO  [22:05:18.740] epoch 2, loss 0.0318
## INFO  [22:05:24.954] epoch 3, loss 0.0267
## INFO  [22:05:31.196] epoch 4, loss 0.0239
## INFO  [22:05:37.341] epoch 5, loss 0.0222
## INFO  [22:05:43.503] epoch 6, loss 0.0209
## INFO  [22:05:49.635] epoch 7, loss 0.0199
## INFO  [22:05:55.737] epoch 8, loss 0.0191
## INFO  [22:06:01.901] epoch 9, loss 0.0184
## INFO  [22:06:08.110] epoch 10, loss 0.0179
```

```r
# Glove model learns two sets of word vectors - main and context.
# Both matrices may be added to get the combined word vector
wv_context <- glove$components
word_vectors_amazon <- wv_main_amazon + t(wv_context)
# Converting the word_vector to a dataframe for visualization
word_vectors_amazon <- data.frame(word_vectors_amazon)
# The word for each embedding is set as row name by default
# Using the tibble library rownames_to_column function, the rownames is copied
# as first column of the dataframe
# We also name the first column of the dataframe as words
library(tibble)
word_vectors_amazon <- rownames_to_column(word_vectors_amazon, var = "words")
```

```r
library(softmaxreg)
docVectors_amazon = function(x) { wordEmbed(x, word_vectors_amazon, meanVec = TRUE) }
# Applying the function docVectors function on the entire reviews dataset
# This will result in word embedding representation of the entire reviews dataset
temp_amazon <- t(sapply(text_amazon$SentimentText, docVectors_amazon))
```

```r
# Splitting the dataset into train and test portions
temp_amazon_train <- temp_amazon[1:N_train_amazon,]
temp_amazon_test <- temp_amazon[(N_train_amazon+1):N_amazon,]
labels_amazon_train <- as.factor(as.character(text_amazon[1:N_train_amazon,]$Sentiment))
labels_amazon_test <- as.factor(as.character(text_amazon[(N_train_amazon+1):N_amazon,]$Sentiment))
# Using randomforest to build a model on train data
library(randomForest)
rf_amazon_senti_classifier <- randomForest(temp_amazon_train, labels_amazon_train,ntree=20)
print(rf_amazon_senti_classifier)
```

```
##
## Call:
##  randomForest(x = temp_amazon_train, y = labels_amazon_train,      ntree = 20)
##                Type of random forest: classification
##                      Number of trees: 20
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 30.7%
## Confusion matrix:
##       1     2 class.error
## 1 27275 11706   0.3003001
## 2 12849 28163   0.3132985
```

```r
# Predicting labels using the randomforest model created
rf_amazon_predicts <- predict(rf_amazon_senti_classifier, temp_amazon_test)
# Estimating the accuracy from the predictions
library(rminer)
print(mmetric(rf_amazon_predicts, labels_amazon_test, c("ACC")))
```

```
## [1] 72.72
```

5.1 Drug Data

```
## INFO  [22:12:42.721] epoch 1, loss 0.0755
## INFO  [22:12:44.912] epoch 2, loss 0.0487
## INFO  [22:12:47.054] epoch 3, loss 0.0401
## INFO  [22:12:49.236] epoch 4, loss 0.0354
## INFO  [22:12:51.421] epoch 5, loss 0.0324
## INFO  [22:12:53.615] epoch 6, loss 0.0302
## INFO  [22:12:55.812] epoch 7, loss 0.0285
## INFO  [22:12:58.014] epoch 8, loss 0.0273
## INFO  [22:13:00.167] epoch 9, loss 0.0262
## INFO  [22:13:02.419] epoch 10, loss 0.0254
```

```
##
## Call:
##  randomForest(x = temp_train_Drug, y = labels_train_Drug, ntree = 20)
##                Type of random forest: classification
##                      Number of trees: 20
## No. of variables tried at each split: 7
##
```

```
##         OOB estimate of  error rate: 29.08%
## Confusion matrix:
##       1     2 class.error
## 1 23380  8604   0.2690095
## 2 10039 22089   0.3124689

## [1] 74.95945
```

6 FastText word embedding

6.1 Amazon reviews

```r
library(fastTextR)
# Input reviews file
text_amazon <- readLines("Sentiment Analysis Dataset_ft.txt")
```

```r
# Dividing the reviews into training and test
temp_amazon_train <- text_amazon[1:N_train_amazon]
temp_amazon_test <- text_amazon[(N_train_amazon+1):N_amazon]
```

```r
# Creating txt file for train and test dataset
fileConn <- file("train.ft.txt")
writeLines(temp_amazon_train, fileConn)
close(fileConn)
fileConn <- file("test.ft.txt")
writeLines(temp_amazon_test, fileConn)
close(fileConn)
# Creating a test file with no labels
temp_amazon_test_nolabel <- gsub("__label__1", "", temp_amazon_test, perl=TRUE)
temp_amazon_test_nolabel <- gsub("__label__2", "", temp_amazon_test_nolabel, perl=TRUE)
```

```r
fileConn <- file("test_nolabel.ft.txt")
writeLines(temp_amazon_test_nolabel, fileConn)
close(fileConn)
# Training a supervised classification model with training dataset file
model_amazon <- ft_train("train.ft.txt", method = "supervised",
                control = ft_control(nthreads = 3L, seed = 1))
# Obtain all the words from a previously trained model
words_amazon <- ft_words(model_amazon)
```

```r
# Obtain word vectors from a previously trained model.
word_vec_amazon <- ft_word_vectors(model_amazon, words_amazon)
```

```r
# Predicting the labels for the reviews in the no labels test dataset
# Getting the predictions into a dataframe so as to compute performance measurement
ft_preds_amazon <- ft_predict(model_amazon, newdata = temp_amazon_test_nolabel)
# Reading the test file to extract the actual labels
reviewstestfile_amazon <- readLines("test.ft.txt")
# Extracting just the labels frm each line
```

```
library(stringi)
actlabels_amazon <- stri_extract_first(reviewstestfile_amazon, regex="\\w+")
# Converting the actual labels and predicted labels into factors
actlabels_amazon <- as.factor(as.character(actlabels_amazon))
ft_preds_amazon <- as.factor(as.character(ft_preds_amazon$label))
# Getting the estimate of the accuracy
library(rminer)
print(mmetric(actlabels_amazon, ft_preds_amazon, c("ACC")))
```

```
## [1] 86.5
```

6.2 Drug Data

```
## [1] 78.665
```

7 Conclusion & Discussion

References

Chinnamgari, S. K. 2019. *R Machine Learning Projects*.

Pennington, Jeffrey, Richard Socher, and Christopher D Manning. 2014. "Glove: Global Vectors for Word Representation." In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1532–43.

Zhang, Xiang, Junbo Zhao, and Yann LeCun. 2015. "Character-Level Convolutional Networks for Text Classification." *Advances in Neural Information Processing Systems* 28.