Report_final

rongl2-xdai12-zixingd2

2022-12-07

Contents

1 Data Prepossessing

1.1 Amazon reviews

```r
set.seed(1)
N <- 100000
N_t <- 0.8*N
reviews_text <- readLines("amazon_review_polarity_csv/train.csv", n = N)
reviews_text <- data.frame(reviews_text)
```

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

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

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

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

```r
print(length(reviews_text))
```

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

```r
# print(head(reviews_text,2))
# Replacing the positive sentiment value 2 with __label__2
reviews_text <- gsub("\\\"2\\\",","__label__2 ",reviews_text)
# Replacing the negative sentiment value 1 with __label__1
reviews_text <- gsub("\\\"1\\\",","__label__1 ",reviews_text)
# Removing the unnecessary \" characters
reviews_text <- gsub("\\\""," ",reviews_text)
# Replacing multiple spaces in the text with single space
reviews_text <- gsub("(?<=[\\s])\\s*|^\\s+|\\s+$", "", reviews_text, perl=TRUE)
# Basic EDA post the required processing to confirm input is as desired
print("EDA POST PROCESSING")
```

```
## [1] "EDA POST PROCESSING"
```

```r
print(class(reviews_text))
```

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

```r
print(length(reviews_text))
```

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

```r
# print(head(reviews_text,2))
# 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_text, fileConn)
close(fileConn)
```

1.2 Drug Data

```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)
```

```
## [1] "character"

## [1] 80150

## [1] "EDA POST PROCESSING"

## [1] "character"

## [1] 80150
```

2 BoW approach

2.1 Amazon reviews

```
library(SnowballC)
library(tm)
# Reading the transformed file as a dataframe
text <- read.table(file='Sentiment Analysis Dataset.csv', sep=',', header = TRUE)
# Checking the dataframe to confirm everything is in tact
print(dim(text))
```

```
## [1] 100000      2
```

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

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 100000
```

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

```
## <<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
```

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

```
## <<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
```

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

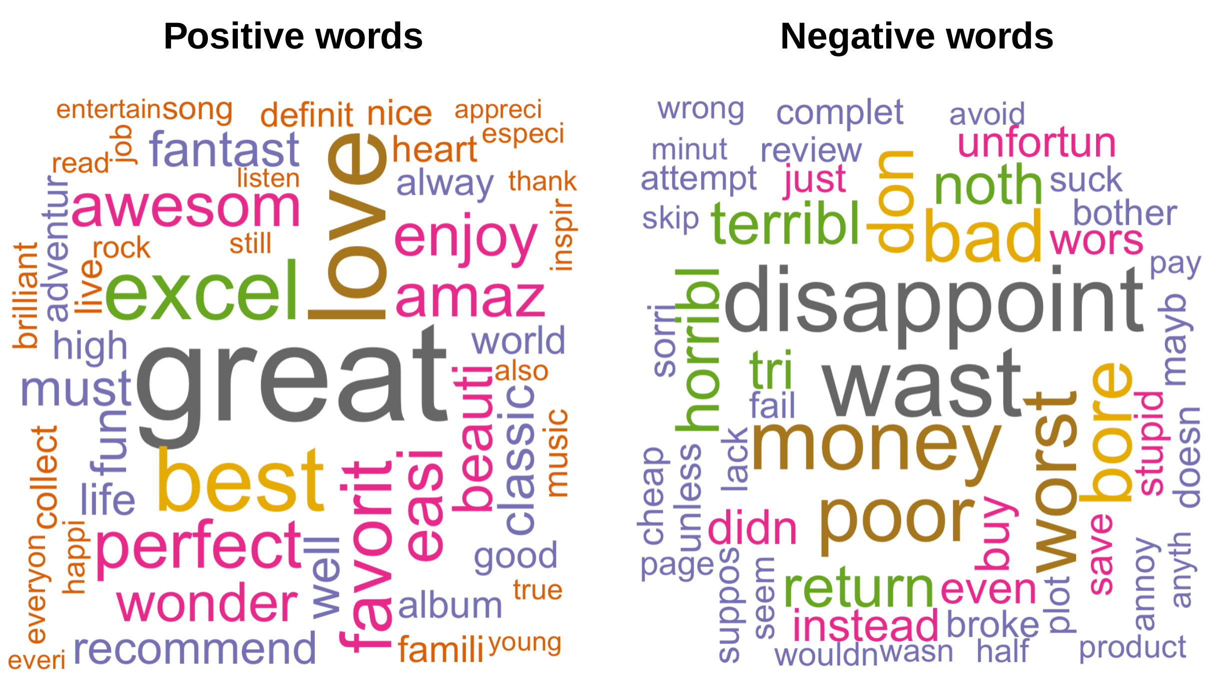
```r
# Using two-sample t-test to find the most different word to show our Word Cloud
library(slam)
summ = matrix(0, nrow=v.size, ncol=4)
summ[,1] = colapply_simple_triplet_matrix(
  as.simple_triplet_matrix(dtm_train[ytrain==2, ]), mean)
summ[,2] = colapply_simple_triplet_matrix(
  as.simple_triplet_matrix(dtm_train[ytrain==2,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

```r
words = colnames(dtm_train)
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]
```

```r
# 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"))
```

```r
# 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"))
```

```r
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)
```



Word Clouds from Amazon reviews

```r
# Splitting the train and test DTM
dtm_train_train <- dtm_train[1:N_t, ]
dtm_train_test <- dtm_train[(N_t+1):N, ]
dtm_train_train_labels <- as.factor(as.character(text[1:N_t, ]$Sentiment))
dtm_train_test_labels <- as.factor(as.character(text[(N_t+1):N, ]$Sentiment))
```

```r
# 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") }
```

```r
# Applying the function to rows in training and test datasets
dtm_train_train <- apply(dtm_train_train, MARGIN = 2,cellconvert)
dtm_train_test <- apply(dtm_train_test, MARGIN = 2,cellconvert)
```

```r
# Training the naive bayes classifier on the training dtm
library(e1071)
nb_senti_classifier <- naiveBayes(dtm_train_train,dtm_train_train_labels)
# Printing the summary of the model created
summary(nb_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
```

```r
# Making predictions on the test data dtm
nb_predicts <- predict(nb_senti_classifier, dtm_train_test,type="class")
```

```r
# Computing accuracy of the model
library(rminer)
print(mmetric(nb_predicts, dtm_train_test_labels, c("ACC")))
```

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

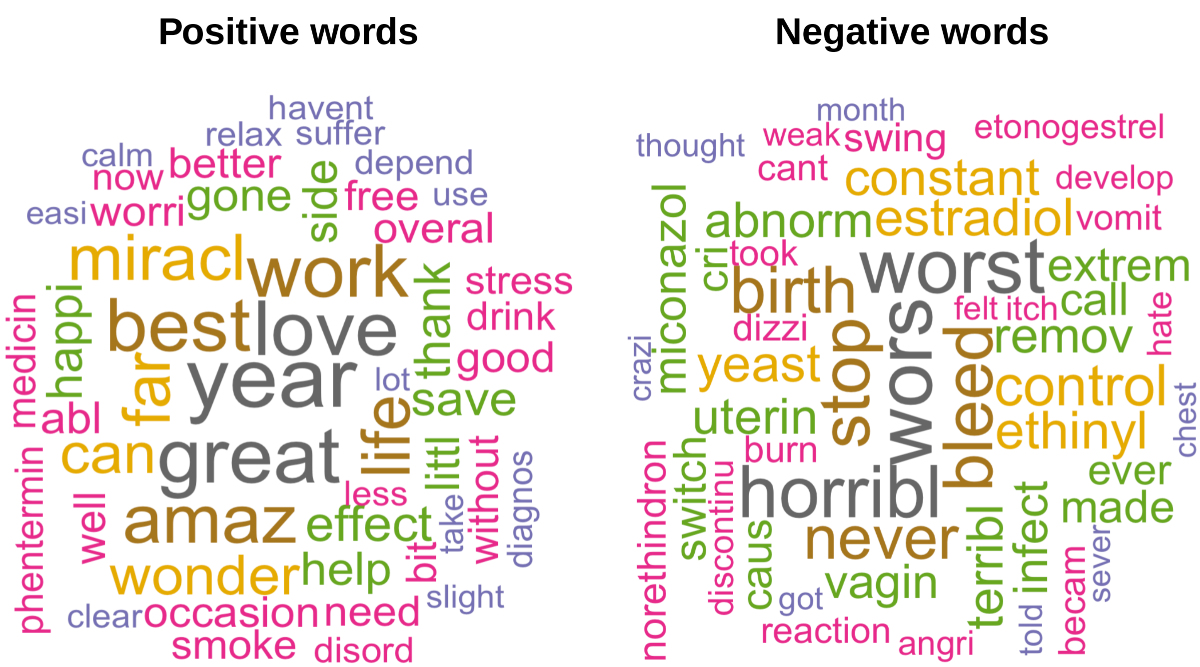
2.2 Drug Data

```
## [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
```

3 Pretrained word2vec word embedding

3.1 Amazon reviews

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

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

```r
# Function to get word vector for each review
docVectors <- function(x) { wordEmbed(x, word2vec, meanVec = TRUE) }
text <- 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 <- t(sapply(text$SentimentText, docVectors))
dim(temp)
```

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

```r
# Splitting the dataset into train and test
temp_train <- temp[1:N_t,]
temp_test <- temp[(N_t+1):N,]
labels_train <- as.factor(as.character(text[1:N_t,]$Sentiment))
labels_test <- as.factor(as.character(text[(N_t+1):N,]$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_senti_classifier <- randomForest(temp_train, labels_train,ntree=20)
print(rf_senti_classifier)
```

```
##
## Call:
##  randomForest(x = temp_train, y = labels_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_predicts <- predict(rf_senti_classifier, temp_test)
library(rminer)
print(mmetric(rf_predicts, labels_test, c("ACC")))
```

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

3.2 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
```

4 GloVe word embedding

4.1 Amazon reviews

```
# Including the required library
library(text2vec)
# Reading the dataset
text <- read.csv(file='Sentiment Analysis Dataset.csv', header = TRUE)
# Subsetting only the review text so as to create Glove word embedding
wiki <- as.character(text$SentimentText)
# Create iterator over tokens
tokens <- space_tokenizer(wiki)
# Create vocabulary. Terms will be unigrams (simple words).
it <- itoken(tokens, progressbar = FALSE)
vocab <- create_vocabulary(it)
# Consider a term in the vocabulary if and only if the term has appeared at least
# three times in the dataset
vocab <- prune_vocabulary(vocab, term_count_min = 3L)
# Use the filtered vocabulary
vectorizer <- vocab_vectorizer(vocab)
```

```r
# Use window of 5 for context words and create a term co-occurance matrix
tcm <- create_tcm(it, vectorizer, 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$new(rank = 50, x_max = 100)
wv_main <- glove$fit_transform(tcm, n_iter = 10, convergence_tol = 0.01)
```

```
## INFO  [02:44:43.472] epoch 1, loss 0.0502
## INFO  [02:44:49.228] epoch 2, loss 0.0318
## INFO  [02:44:54.991] epoch 3, loss 0.0267
## INFO  [02:45:00.853] epoch 4, loss 0.0239
## INFO  [02:45:06.727] epoch 5, loss 0.0222
## INFO  [02:45:12.511] epoch 6, loss 0.0209
## INFO  [02:45:18.310] epoch 7, loss 0.0199
## INFO  [02:45:24.208] epoch 8, loss 0.0191
## INFO  [02:45:30.008] epoch 9, loss 0.0184
## INFO  [02:45:36.338] 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 <- wv_main + t(wv_context)
# Converting the word_vector to a dataframe for visualization
word_vectors <- data.frame(word_vectors)
# 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 <- rownames_to_column(word_vectors, var = "words")
```

```r
library(softmaxreg)
docVectors = function(x) { wordEmbed(x, word_vectors, 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 <- t(sapply(text$SentimentText, docVectors))
```

```r
# Splitting the dataset into train and test portions
temp_train <- temp[1:N_t,]
temp_test <- temp[(N_t+1):N,]
labels_train <- as.factor(as.character(text[1:N_t,]$Sentiment))
labels_test <- as.factor(as.character(text[(N_t+1):N,]$Sentiment))
# Using randomforest to build a model on train data
library(randomForest)
rf_senti_classifier <- randomForest(temp_train, labels_train,ntree=20)
print(rf_senti_classifier)
```

```
##
## Call:
##  randomForest(x = temp_train, y = labels_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
```

```
# Predicting labels using the randomforest model created
rf_predicts <- predict(rf_senti_classifier, temp_test)
# Estimating the accuracy from the predictions
library(rminer)
print(mmetric(rf_predicts, labels_test, c("ACC")))
```

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

4.2 Drug Data

```
## INFO  [02:51:36.401] epoch 1, loss 0.0755
## INFO  [02:51:38.728] epoch 2, loss 0.0487
## INFO  [02:51:41.112] epoch 3, loss 0.0401
## INFO  [02:51:43.606] epoch 4, loss 0.0354
## INFO  [02:51:45.960] epoch 5, loss 0.0324
## INFO  [02:51:48.300] epoch 6, loss 0.0302
## INFO  [02:51:50.673] epoch 7, loss 0.0285
## INFO  [02:51:53.048] epoch 8, loss 0.0273
## INFO  [02:51:55.453] epoch 9, loss 0.0262
## INFO  [02:51:57.799] 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
```

5 FastText word embedding

5.1 Amazon reviews

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

```r
# Dividing the reviews into training and test
temp_train <- text[1:N_t]
temp_test <- text[(N_t+1):N]
```

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

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

```r
# Obtain word vectors from a previously trained model.
word_vec <- ft_word_vectors(model, words)
```

```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 <- ft_predict(model, newdata = temp_test_nolabel)
# Reading the test file to extract the actual labels
reviewstestfile <- readLines("test.ft.txt")
# Extracting just the labels frm each line
library(stringi)
actlabels <- stri_extract_first(reviewstestfile, regex="\\w+")
# Converting the actual labels and predicted labels into factors
actlabels <- as.factor(as.character(actlabels))
ft_preds <- as.factor(as.character(ft_preds$label))
# Getting the estimate of the accuracy
library(rminer)
print(mmetric(actlabels, ft_preds, c("ACC")))
```

```
## [1] 86.515
```

5.2 Drug Data

```
## [1] 78.70867
```