TextMinning_Modelling_Project4

Samriti Malhotra & Vishal Arora

April 13, 2019

Contents

Introduction

As part of Project 4 we are trying to classify new "test" documents using already classified "training" documents. A common example is using a corpus of labeled spam and ham (non-spam) e-mails to predict whether or not a new document is spam.

For this project, we used spam/ham dataset, then predict the class of new documents (either withheld from the training dataset or from another source such as your own spam folder). It can be useful to be able to classify new "test" documents using already classified "training" documents. A common example is using a corpus of labeled spam and ham (non-spam) e-mails to predict whether or not a new document is spam. Example corpus is taken from path: -Old Apache spam ham link for download

Problem Statment

Download the spam/ham file and write them into csv file, which has two columns text and spam indicator.

Solution

Step1:-

In our soltuion is to get a common csv file constructed from the spam and hard ham files ,We wrote a function which will take 3 parameteres , filename, where to construct the csv file and column to be added to final csv file ie. spam=TRUE for email which are span and spam=FALSE for emails which are not spam.

```
txt2csv <- function(myfiles, mycsvfilename , indicator  ){

starting_dir <- getwd()
# create a list of dataframes containing the text
```

```
mytxts <- lapply(myfiles, readLines)
# combine the rows of each dataframe to make one
# long character vector where each item in the vector
# is a single text file
mytxts1lines <- unlist(mytxts)
mytxts1lines <- paste(mytxts1lines , collapse = ",")
mytxts1lines <- str_replace_all(mytxts1lines, ",", " ")
# make a dataframe with the file names and texts
mytxtsdf <- data.frame(filename = basename(myfiles),fulltext = mytxts1lines)
#dropping the first column and  binding column wise using cbinf.
mytxtsdf <- cbind(dplyr::select(mytxtsdf,-1),indicator)

  # Now write them all into a single CSV file, one txt file per row
  setwd(mydir) # make sure the CSV goes into the dir where the txt files are

    # write the CSV file...
    write.table(mytxtsdf, file = paste0(mycsvfilename, ".csv"),sep=",", row.names = FALSE, col.names = 

  # return original working directory
  setwd(starting_dir)
}
setwd("C:/CUNY_AUG27/Data607/Chapter10/corpus")
mydir <- "C:/CUNY_AUG27/Data607/Chapter10/corpus/spam_2/"
myfiles <- list.files(mydir, full.names = TRUE)
getwd()
for(i in 1:length(myfiles)){
  print(myfiles[i])
  txt2csv(myfiles[i], "C:/CUNY_AUG27/Data607/Chapter10/datacsvfile" , TRUE)
}
mydir <- "C:/CUNY_AUG27/Data607/Chapter10/corpus/hard_ham/"
myfiles <- list.files(mydir, full.names = TRUE)
for(i in 1:length(myfiles)){
  #print(myfiles[i])
  txt2csv(myfiles[i], "C:/CUNY_AUG27/Data607/Chapter10/datacsvfile" , FALSE)
}
```

Steps2:-

This section , has been commented if you want to run in your local set echo=TRUE chunk parameters and run to construct the csv file , and also if by any chance you want to build again then please delete the old one.

```
## 'data.frame':    764 obs. of  2 variables:
##  $ text     : chr  "From lmrn@mailexcite.com  Mon Jun 24 17:03:24 2002 Return-Path: merchantsworld20
##  $ spamIndic: logi  TRUE TRUE TRUE TRUE TRUE TRUE ...
```

x
From lmrn@mailexcite.com Mon Jun 24 17:03:24 2002 Return-Path:
merchantsworld2001@juno.com Delivery-Date: Mon May 13 04:46:13
2002 Received: from mandark.labs.netnoteinc.com
([213.105.180.140]) by dogma.slashnull.org (8.11.6/8.11.6) with
ESMTP id g4D3kCe15097 for <jm@jmason.org>; Mon 13 May 2002
04:46:12 +0100 Received: from

x
TRUE

```
## [1] 591930
```

```
## [1] 0
```

```
## [1] 760
```

```
## [1] 480
```

Var1	Freq
FALSE	132
TRUE	631

##Step3 :- Creating a corpus
Follow the standard steps to build and pre-process the corpus:
1) Build a new corpus variable called corpus.
2) Using tm__map, convert the text to lowercase.
3) Using tm__map, remove all punctuation from the corpus.
4) Using tm__map, remove all English stopwords from the corpus.
5) Using tm__map, stem the words in the corpus.
6) Build a document term matrix from the corpus, called dtm.

Step4:-

Further pre-processing of the corpus,Removing stop words.emoving words can be done with the removeWords argument to the tm_map() function, with an extra argument, i.e. what the stop words are that we want to remove, for which we simply use the list for english that is provided by the tm package.

Step5:-

Stemming :-
Lastly, we want to stem our document with the stemDocument argument.Stemming is nothing but collapsing/removing words which have same root.

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

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

Step6:-

BAG OF WORDS
Create a Document Term Matrix
We are now ready to extract the word frequencies to be used in our prediction problem. The tm package provides a function called DocumentTermMatrix() that generates a matrix where:
–the rows correspond to documents,and
–the columns correspond to words .
–The values in the matrix are the number of times that word appears in each document.

```
tdm = DocumentTermMatrix(corpus)
#kable(head(tdm)) %>%
#  kable_styling(bootstrap_options = c("striped","condensed","responsive"),full_width   = F,position =
#  row_spec(0, background ="gray")
```

Step7:-

Using sparseRemove to remove the terms which are not often appear or they don't create any additional benefits but removing them adds to computation benefits,to obtain a more reasonable number of terms, limit dtm to contain terms appearing in at least 5% of documents.

Remove sparse terms.

	x
yyyylocalhostspamassassintaintorg	41
namedsentto	43
jmasonorg	43
namedcontactnam	44
actiondhttpresponseresponseasp	46
namedbtnsubmit	46

Now let's see how many time word stems appear in the ham emails in the dataset We can read the most frequent terms in the ham dataset and the spam subsets.

```
tail(sort(colSums(subset(emailsSparse, spam == FALSE))))
```

```
##   font widthd border   tabl height  width
##   2956   2986   3879   4453   4713  11606
```

```
#kable(tail(sort(colSums(subset(emailsSparse, spam == FALSE)))) %>%
#  kable_styling(bootstrap_options = c("striped","condensed","responsive"),full_width   = F,position =
#  row_spec(0, background ="gray")


head(sort(colSums(subset(emailsSparse, spam == TRUE))))
```

```
##               cnet         abvsfoacag abvsfoacmtacnetcom
##                  0                  0                  0
##           accucast      bgcolorffffef           divtdtrtr
##                  0                  0                  0
```

```
#kable(head(sort(colSums(subset(emailsSparse, spam == TRUE))))) %>%
#  kable_styling(bootstrap_options = c("striped","condensed","responsive"),full_width   = F,position =
#  row_spec(0, background ="gray")
```

Step8 :-

Build Machine learnig model (SVM i.e. Support Vector Machine)
First, convert the dependent variable to a factor .
Before building the model we need to split our data into training and testing by using sample.split function with 70 data in training and rest in test.
Train and Test Subset, use the subset function TRUE for the train and FALSE for the test.
After training the data, testing the data against the test data to validate out model and ascertain the accuracy of our Model(SVM).

```
## [1]  763 1303
```

```
## [1] "Percentage: 66.58 %"
```

```
## [1] "Percentage: 33.42 %"
```

```
##
```

```
## Call:
## svm(formula = emailsSparse.train$spam ~ ., data = emailsSparse.train,
##     method = "class")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.0007680492
##
## Number of Support Vectors:  205
##
##  ( 125 80 )
##
##
## Number of Classes:  2
##
## Levels:
##  FALSE TRUE
```

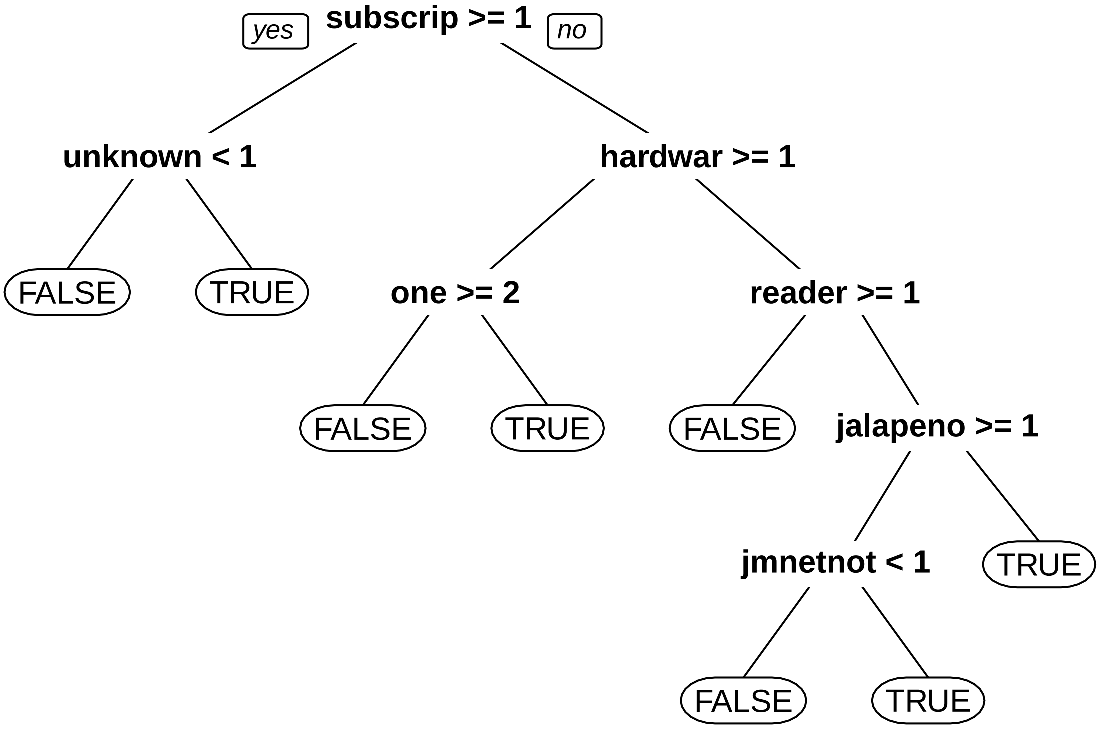
```
##             Predicted Class
## Actual Class FALSE TRUE
##        FALSE    33   17
##        TRUE     12  193
```

```
## [1] 0.8941176
```

Step9:-

BUILD A CART MODEL (Decision Tree)
Plotting the decission tree at the end.

```
spamCART = rpart(emailsSparse.train$spam~., data=emailsSparse.train, method="class")
prp(spamCART)
```



```r
predTestCART = predict(spamCART, newdata=emailsSparse.test)[,2]

#What is the testing set AUC of spamCART?
#predictionTestCART = prediction(predTestCART, emailsSparse.test$spam)
#as.numeric(performance(predictionTestCART, "auc")@y.values)

#What is the testing set accuracy of spamRF, using a threshold of 0.5 for predictions?

table(emailsSparse.test$spam, predTestCART  > 0.5)
```

```
##
##          FALSE TRUE
##    FALSE    38   12
##    TRUE     11  194
```

```r
#Accuracy
(20+208)/nrow(emailsSparse.test)
```

```
## [1] 0.8941176
```

Step10:-

BUILD A RANDOM FOREST
We are creating one more model random forest model and called this model as spamRF. Create Random Forest Model using all the parameters.

```r
spamRF = randomForest(emailsSparse.train$spam~., data=emailsSparse.train)
predTrainRF = predict(spamRF, type="prob")[,2]
```

```
#What is the training set accuracy of spamRF, using a threshold of 0.5 for predictions?

#table(emailsSparse.train$spam, predTrainRF > 0.5)
#Accuracy
#(76+412)/nrow(emailsSparse.test)
```

Out-of-Sample Performance of the above model(Random Forest) we created Now that we have trained a models, we need to evaluate it on the test set, and also calc ulate the accuracy of each model.

```
#Accuracy of RAndom Forest model against test data
predTestRF = predict(spamRF, newdata=emailsSparse.test, type="prob")[,2]
table(emailsSparse.test$spam, predTestRF > 0.5)
```

```
##
##         FALSE TRUE
##   FALSE    37   13
##   TRUE      3  202
```

```
#Accuracy
(31+207)/nrow(emailsSparse.test)
```

```
## [1] 0.9333333
```

```
#predictionTestRF = prediction(predTestRF, emailsSparse.test$spam)
#as.numeric(performance(predictionTestRF, "auc")@y.values)
```

Summary

Looking at above models implemented for our test dataset, and the accuracy predicted for all the three models it seems that Random Forest Model's accuracy(93%) is more accurate than CART Model(Decision Model 89%) & SVM (89% Support Vector Machine) Model.