Data-607 Project-4

Student Name : Sachid Deshmukh

Date : 11/02/2018

- GitHub Location for rmd file
- GitHub Location for pdf file
- RPubs location of published file

Let's briefly review some of the steps in a typical text analysis pipeline:

1] Construct a document-term matrix (DTM). In other words, the first step is to vectorize text by creating a map from words or n-grams to a vector space.

2] Fit a model to that DTM. These models might include text classification, topic modeling, similarity search, etc. Fitting the model will include tuning and validating the model.

3] Use train model for prediction

Following packages are required for this project

- text2vec
- data.table
- magrittr
- caTools
- glmnet
- caret

```
library(text2vec)
library(data.table)
library(magrittr)
library(caTools)
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-5
```

```
library(caret)
```

```
## Loading required package: ggplot2
```

Utility function section

```
ReadInput = function(DirPath, DocType)
{
  Files = list.files(DirPath)
```

```
  FileList = NULL
  for(i in 1:length(Files))
  {
    FilePath <-paste0(DirPath, "/", Files[1])
    FileText <-readLines(FilePath)
    if(!is.na(FileText) && length(FileText) > 0)
    TempList <- list(paste(FileText, collapse="\n"))
    FileList = c(FileList,TempList)

  }
  DocDF <-as.data.frame(unlist(FileList),stringsAsFactors = FALSE)
  DocDF$Type <- ifelse(DocType == "Spam",1, 0)
  colnames(DocDF) <- c("Text","Type")
  return(DocDF)
}

CreateDocumentTermMatrix = function(DataInput)
{
  PreProcess.Fun = tolower
  Tokenizing.Fun = word_tokenizer

  Tokenizer = itoken(DataInput$Text,
             preprocessor = PreProcess.Fun,
             tokenizer = Tokenizing.Fun,
             ids = DataInput$Id,
             progressbar = FALSE)
  Document.Vocab = create_vocabulary(Tokenizer)
  Vectorizer = vocab_vectorizer(Document.Vocab)
  Doc.Term.Matrix = create_dtm(Tokenizer, Vectorizer)
  return(Doc.Term.Matrix)
}
```

Load data and split into train and test parts

First of all let's split out dataset into two parts - train and test.

```
# Create Input Data frame
Ham.DF = ReadInput("D:/MSDS/ExternalData/Ham", "Ham")
Spam.DF = ReadInput("D:/MSDS/ExternalData/Spam/", "Spam")
Doc.DF <- rbind(Ham.DF, Spam.DF)
Doc.DF = cbind(Id=rownames(Doc.DF), Doc.DF)
set.seed(123)
sample = sample.split(Doc.DF$Id, SplitRatio = 0.70)
Train.Doc.DF = subset(Doc.DF, sample==TRUE)
Test.Doc.DF = subset(Doc.DF, sample==FALSE)
```

Create Document term matrix for train and test data

To represent documents in vector space, we first have to create mappings from terms to term IDS. We call them terms instead of words because they can be arbitrary n-grams not just single words. We represent a set of documents as a sparse matrix, where each row corresponds to a document and each column corresponds to a term. This can be done in 2 ways: using the vocabulary itself or by feature hashing.

```
Train.DTM = CreateDocumentTermMatrix(Train.Doc.DF)
Test.DTM = CreateDocumentTermMatrix(Test.Doc.DF)
```

Train the model

Now we are ready to fit our first model. Here we will use the glmnet package to fit a logistic regression model with an L1 penalty and 10 fold cross-validation.

```
# Train Model

NFOLDS = 10
t1 = Sys.time()
glmnet_classifier = cv.glmnet(x = Train.DTM, y = Train.Doc.DF$Type,
                              family = 'binomial',
                              # L1 penalty
                              alpha = 1,
                              # interested in the area under ROC curve
                              type.measure = "auc",
                              # 5-fold cross-validation
                              nfolds = NFOLDS,
                              # high value is less accurate, but has faster training
                              thresh = 1e-3,
                              # again lower number of iterations for faster training
                              maxit = 1e3)
```

Make predictions using train model

We have successfully fit a model to our DTM. Now we can check the model's performance on test data. Note that we use exactly the same functions from prepossessing and tokenization. Also we reuse/use the same vectorizer - function which maps terms to indices.

Imp Note : By examining the output predictions and cross validation results it looks like we can select 0.15 as cutoff for Spam/Ham classification

```
preds = predict(glmnet_classifier, Test.DTM, type = 'response')
preds = factor(ifelse(preds <= 0.15, 0, 1))
```

Print out Confusion Matrix

From the below confusion matrix we can see that we have 100% accuracy. We have 100% True Positives and 100% False Positives Model also is doing very good job of avoiding False Positives and False Negative which are 0%. These stats are great!!!

```
confusionMatrix(preds, factor(Test.Doc.DF$Type), positive = NULL, dnn = c("Prediction","Actual"))
```

```
## Confusion Matrix and Statistics
##
##            Actual
## Prediction   0   1
##          0 758   0
##          1   0 158
##
##                Accuracy : 1
```

```
##                  95% CI : (0.996, 1)
##     No Information Rate : 0.8275
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.8275
##          Detection Rate : 0.8275
##    Detection Prevalence : 0.8275
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : 0
##
```

- Referenced Material