Spam or Ham Classfication Project

Vinoth Aryan Nagabosshanam

March 26, 2017

Abstract

Build a machine learning model to predicted accurately classify which texts are spam or ham

Data Description

*** The dataset can be downloaded at http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/, through the UCI Machine Learning Repository or through Kaggle.***

The Data contain one message per line. Each line is composed by two columns:

v1: contains the label (ham or spam) v2 : contains the raw text

** import the data file **

```
sms_data<-read.csv("B:\\Spam_or_Ham\\sms-spam-collection-dataset\\spam.csv")
str(sms_data)

## 'data.frame':    5572 obs. of  5 variables:
##  $ v1 : Factor w/ 2 levels "ham","spam": 1 1 2 1 1 2 1 1 2 2 ...
##  $ v2 : Factor w/ 5169 levels "'An Amazing Quote'' - \\Sometimes in life
its difficult to decide whats wrong!! a lie that brings a smile or the truth
that bri"| __truncated__,..: 1160 3261 1061 4281 2910 1086 994 433 4764 1294
...
```

** there total five variables but we are going to use only the two variables V1 and v2 and remove the remaining three variables**

The following code used to remove the three colunms

```
sms_data<-sms_data[ ,-c(3:5)]
# rechange the colunm names
colnames(sms_data)<-c("label","text")
str(sms_data)

## 'data.frame':    5572 obs. of  2 variables:
##  $ label: Factor w/ 2 levels "ham","spam": 1 1 2 1 1 2 1 1 2 2 ...
##  $ text : Factor w/ 5169 levels "'An Amazing Quote'' - \\Sometimes in life
its difficult to decide whats wrong!! a lie that brings a smile or the truth
```

```
that bri"| __truncated__,..: 1160 3261 1061 4281 2910 1086 994 433 4764 1294
...

# step to convert the label as factore
sms_data$label<-as.factor(sms_data$label)
summary(sms_data$label)

##  ham spam
## 4825  747
```

Data Preprocessing

Building a corpus

Let’s now build a corpus out of this vector of strings. A corpus is a collection of documents, but it’s also important to know that in the tm domain, R recognizes it as a separate data type.

There are two kinds of the corpus data type, the permanent corpus, i.e. PCorpus, and the volatile corpus, i.e. VCorpus. In essence, the difference between the two has to do with how the collection of documents is stored in your computer. We will use the volatile corpus, which is held in computer’s RAM rather than saved to disk, just to be more memory efficient.

To make a volatile corpus, R needs to interpret each element in our vector of text, text, as a document. And the tm package provides what are called Source functions to do just that! In this exercise, we’ll use a Source function called vectorSource() because our text data is contained in a vector. The output of this function is called a Source object.

```
## Warning: package 'tm' was built under R version 3.3.3

## Warning: package 'NLP' was built under R version 3.3.3
```

Now that we’ve converted our vector to a Source object, we pass it to another tm function, VCorpus(), to create our volatile corpus. The VCorpus object is a nested list, or list of lists. At each index of the VCorpus object, there is a PlainTextDocument object, which is essentially a list that contains the actual text data (content), as well as some corresponding metadata (meta) which can help to visualize a VCorpus object and to conceptualize the whole thing.

```
# Make a volatile corpus: sms_corpus
sms_corpus <- VCorpus(sms_cor)
# Print out the sms_corpus
sms_corpus

## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 5572
```

```r
sms_data$label[1:4]

## [1] ham  ham  spam ham
## Levels: ham spam

# Check the text in some messages and their type
lapply(sms_corpus[1:4], as.character)

## $`1`
## [1] "Go until jurong point, crazy.. Available only in bugis n great world
la e buffet... Cine there got amore wat..."
##
## $`2`
## [1] "Ok lar... Joking wif u oni..."
##
## $`3`
## [1] "Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005.
Text FA to 87121 to receive entry question(std txt rate)T&C's apply
08452810075over18's"
##
## $`4`
## [1] "U dun say so early hor... U c already then say..."

sms_corpus[[23]][1]

## $content
## [1] "So Ì_ pay first lar... Then when is da stock comin..."

sms_data$label[23]

## [1] ham
## Levels: ham spam
```

Cleaning and preprocessing of the text

After obtaining the corpus, usually, the next step will be cleaning and preprocessing of the text. For this endeavor we are mostly going to use functions from the tm and qdap packages. In bag of words text mining, cleaning helps aggregate terms. For example, it may make sense that the words “miner”, “mining” and “mine” should be considered one term. Specific preprocessing steps will vary based on the project. For example, the words used in tweets are vastly different than those used in legal documents, so the cleaning process can also be quite different.

Common preprocessing functions include:

tolower(): Make all characters lowercase

removePunctuation(): Remove all punctuation marks

removeNumbers(): Remove numbers

stripWhitespace(): Remove excess whitespace

tolower() is part of base R, while the other three functions come from the tm package.

Making a document-term matrix

The document-term matrix is used when you want to have each document represented as a row. This can be useful if you are comparing authors within rows, or the data is arranged chronologically and you want to preserve the time series.

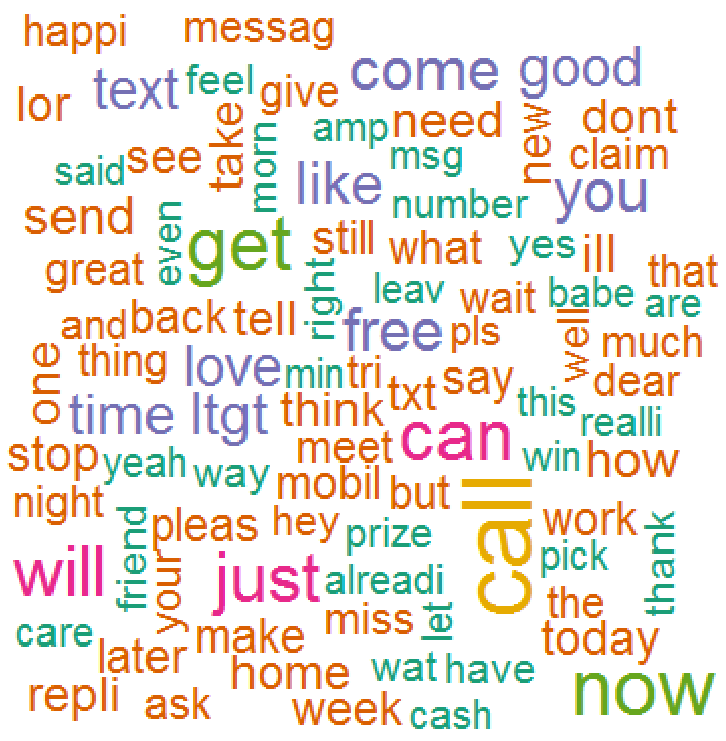
```
# Create Document Term Matrix
d_sms <- DocumentTermMatrix(corpus_clean)

d_sms

## <<DocumentTermMatrix (documents: 5572, terms: 6681)>>
## Non-/sparse entries: 45071/37181461
## Sparsity           : 100%
## Maximal term length: 40
## Weighting          : term frequency (tf)
```

Data visualization

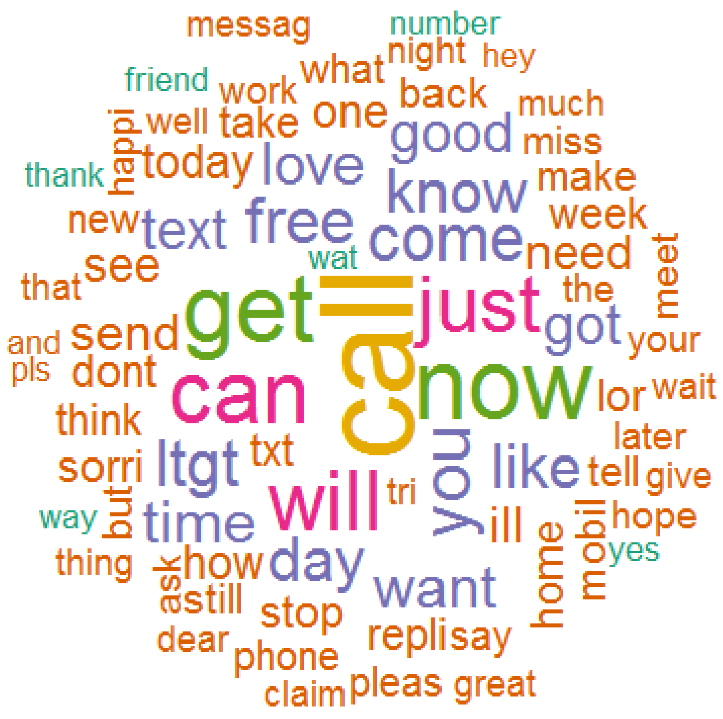
```
{ r echo=FALSE, results='hide',message=FALSE} #library(plotly) #b<-
barchart(sms_data$label,horizontal=F,col=c('red','green'),ylab='count',main='
Barchart') #b
```

We can easily create a wordcloud by using the wordcloud() function from the wordcloud package

```
library(wordcloud)

## Warning: package 'wordcloud' was built under R version 3.3.3

## Loading required package: RColorBrewer

wordcloud(corpus_clean,
max.words=100,scale=c(3,1),colors=brewer.pal(6,"Dark2"))

## Warning in wordcloud(corpus_clean, max.words = 100, scale = c(3, 1),
colors
## = brewer.pal(6, : sorri could not be fit on page. It will not be plotted.
## Warning in wordcloud(corpus_clean, max.words = 100, scale = c(3, 1),
colors
## = brewer.pal(6, : hope could not be fit on page. It will not be plotted.

## Warning in wordcloud(corpus_clean, max.words = 100, scale = c(3, 1),
colors
## = brewer.pal(6, : phone could not be fit on page. It will not be plotted.

## Warning in wordcloud(corpus_clean, max.words = 100, scale = c(3, 1),
colors
## = brewer.pal(6, : day could not be fit on page. It will not be plotted.

## Warning in wordcloud(corpus_clean, max.words = 100, scale = c(3, 1),
colors
## = brewer.pal(6, : know could not be fit on page. It will not be plotted.

## Warning in wordcloud(corpus_clean, max.words = 100, scale = c(3, 1),
colors
## = brewer.pal(6, : want could not be fit on page. It will not be plotted.

## Warning in wordcloud(corpus_clean, max.words = 100, scale = c(3, 1),
colors
## = brewer.pal(6, : got could not be fit on page. It will not be plotted.

## Warning in wordcloud(corpus_clean, max.words = 100, scale = c(3, 1),
## colors = brewer.pal(6, : tomorrow could not be fit on page. It will not be
## plotted.
```

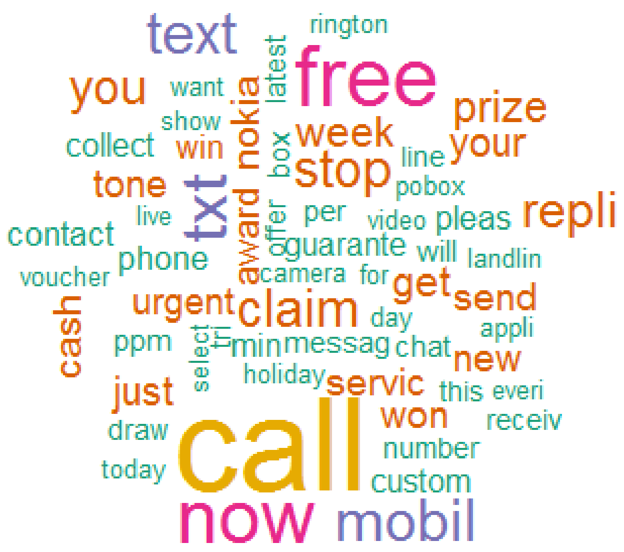


```
wordcloud(corpus_clean,
          min.freq = 100,
          random.order = FALSE,
          colors = brewer.pal(6,"Dark2")
          )
```



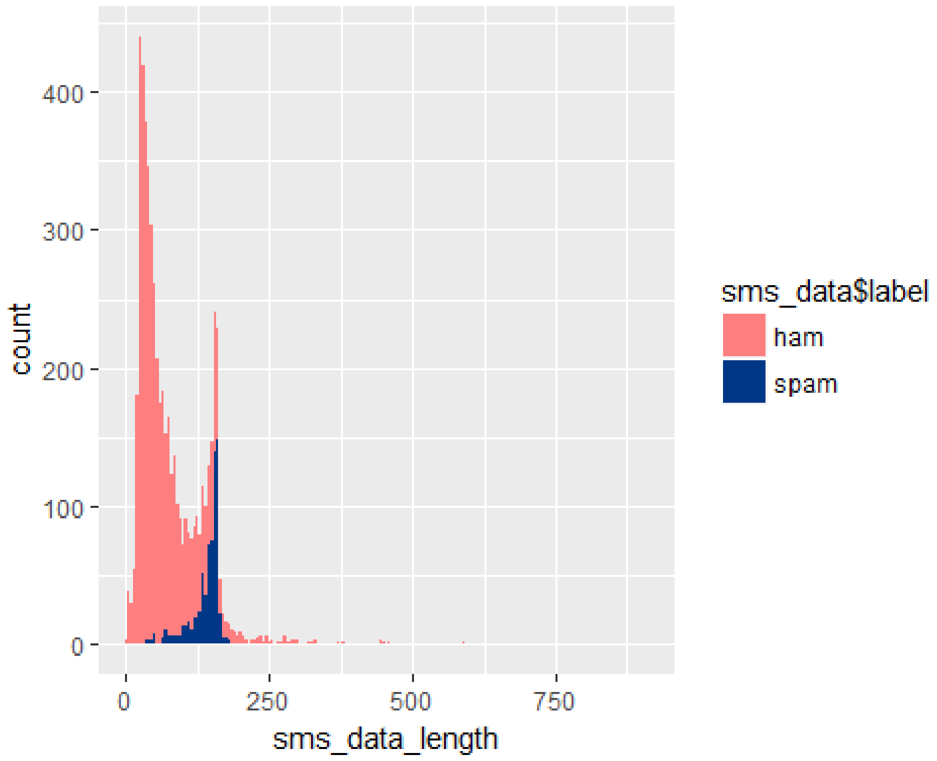
```
library(readr)

## Warning: package 'readr' was built under R version 3.3.3

# look at words that appear atleast 200 times
findFreqTerms(d_sms, lowfreq = 200)

##  [1] "call" "can"  "come" "day"  "free" "get"  "good" "got"  "ill"  "just"
## [11] "know" "like" "love" "ltgt" "now"  "send" "text" "time" "want" "will"
## [21] "you"

s_word <- removeSparseTerms(d_sms, 0.995)
s_word

## <<DocumentTermMatrix (documents: 5572, terms: 333)>>
## Non-/sparse entries: 25431/1830045
## Sparsity           : 99%
## Maximal term length: 9
## Weighting          : term frequency (tf)
```

```
#organizing frequency of terms
freqen <- colSums(as.matrix(s_word))
length(freqen)

## [1] 333

wf <- data.frame(word = names(freqen), freq = freqen)
head(wf)

##                word freq
## account     account   43
## actual       actual   34
## afternoon afternoon   28
## aight         aight   33
## all             all   44
## alreadi     alreadi   90

##Let's create the word cloud  spam to understand

spamcloud<- which(sms_data$label=="spam")

wordcloud(corpus_clean[spamcloud],min.freq=30 ,colors =
brewer.pal(6,"Dark2"))
```



Distribution based on SMS -Length

```
library(ggplot2)

## Warning: package 'ggplot2' was built under R version 3.3.3

##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:NLP':
##
##     annotate

library(stringr)
sms_data_length<-str_length(sms_data$text)
summary(sms_data_length)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2.00   36.00   61.00   80.12  121.00  910.00

ggplot(sms_data,aes(x=sms_data_length,fill=sms_data$label))+geom_histogram(bi
nwidth=5)+scale_fill_manual(values=c("#ff7f80","#003787"))+labs("Distribution
based SMS length")
```



forming training and test data

```
s_word <- as.data.frame(as.matrix(s_word))
#str(s_word)
colnames(s_word) <- make.names(colnames(s_word))
s_word$label <- sms_data$label
#s_word$label

### Finding Frequent Terms
freq_6<-findFreqTerms(d_sms,6)
length(freq_6)

## [1] 1281

freq_6[1:10]

##  [1] "�ûò"     "abiola"  "abl"     "about"   "abt"     "accept"  "access"
##  [8] "account" "across"  "activ"
```

Traning and Test Data forming

```
corpus_train<-corpus_clean[1:4150]
corpus_test<-corpus_clean[4151:5572]


spam_dtrain<-d_sms[1:4150,]
spam_dtest<-d_sms[4151:5572,]

spam_dtrain_label<-sms_data[1:4150,]$label
spam_dtest_label<-sms_data[4151:5572,]$label
prop.table(table(spam_dtrain_label))

## spam_dtrain_label
##       ham      spam
## 0.8650602 0.1349398

prop.table(table(spam_dtest_label))

## spam_dtest_label
##       ham      spam
## 0.8684951 0.1315049

dtm_train<- spam_dtrain[, freq_6]
dim(dtm_train)

## [1] 4150 1281

dtm_test<- spam_dtest[,freq_6]
dim(dtm_test)

## [1] 1422 1281
```

Convert numeric values into categorical values

In order to use the Naive Bayes classifier we need to convert the numerical features in our Document Term Matrix (DTM) to categorical features.

We will convert the numeric features by creating a function that converts any non-zero positive value to "Yes" and all zero values to "No" to indicate whether a specific term is present in the document.

```
pre <- function(x) {
  y <- ifelse(x > 0, "yes","no")
    y
}

train<- apply(dtm_train, 2, pre)

test <- apply(dtm_test, 2, pre)
test[1:10,450:456]

##       Terms
## Docs   guid guy  gym  had  haf   haha hai
##   4151 "no" "no" "no" "no" "no"  "no" "no"
##   4152 "no" "no" "no" "no" "no"  "no" "no"
##   4153 "no" "no" "no" "no" "no"  "no" "no"
##   4154 "no" "no" "no" "no" "yes" "no" "no"
##   4155 "no" "no" "no" "no" "no"  "no" "no"
##   4156 "no" "no" "no" "no" "no"  "no" "no"
##   4157 "no" "no" "no" "no" "no"  "no" "no"
##   4158 "no" "no" "no" "no" "no"  "no" "no"
##   4159 "no" "no" "no" "no" "no"  "no" "no"
##   4160 "no" "no" "no" "no" "no"  "no" "no"
```

buliding a model using Naive bayes

```
library(e1071)

## Warning: package 'e1071' was built under R version 3.3.3

set.seed(12345)
spam_ham_classifier <- naiveBayes(train,spam_dtrain_label)



pred <- predict(spam_ham_classifier, test)
```

Confusion Matrix

```
library(caret)
conf<- confusionMatrix(pred, spam_dtest_label)

conf

confusion_matrix <- as.data.frame(table(pred, spam_dtest_label))

print("the accuracy of this model is 97%")
```

```
Confusion Matrix and Statistics

          Reference
Prediction  ham spam
      ham  1227   22
      spam    8  165

               Accuracy : 0.9789
                 95% CI : (0.97, 0.9857)
    No Information Rate : 0.8685
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.9046
 Mcnemar's Test P-Value : 0.01762
```