Spam or Ham

Sara Parizi

Contents

1 Read_in data

```
# reading the csv file
sms_raw = read.csv("sms_spam.csv", header = TRUE)
```

2 Explore the data

```
# exploring the data structure for each column
str(sms_raw)
```

```
## 'data.frame':    5559 obs. of  2 variables:
##  $ type: chr  "ham" "ham" "ham" "spam" ...
##  $ text: chr  "Hope you are having a good week. Just checking in" "K..give back my th
```

2.1 Converting type to factor

```
# converting the class column (type) to factor
sms_raw$type = factor(sms_raw$type)
str(sms_raw)
```

```
## 'data.frame':    5559 obs. of  2 variables:
##  $ type: Factor w/ 2 levels "ham","spam": 1 1 1 2 2 1 1 1 2 1 ...
##  $ text: chr  "Hope you are having a good week. Just checking in" "K..give back my th
```

2.2 How many ham and spam

```
table(sms_raw$type)
```

```
##
##  ham spam
## 4812  747
```

2.3 Percentage of each type

```
round(prop.table(table(sms_raw$type)) * 100, 1)
```

```
##
##  ham spam
## 86.6 13.4
```

3 Data Preparation - Cleaning and standardizing text data

```
# install.packages("tm")
library(tm)
```

```
## Loading required package: NLP
```

```
library(NLP)
```

3.1 Create the text corpus

```
# Creating the text corpus
sms_corpus = VCorpus(VectorSource(sms_raw$text))
```

3.1.1 Examine the sms corpus

```
inspect(sms_corpus[1:2])
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 2
##
## [[1]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 49
##
## [[2]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 23
```

3.1.2 To see the actual message

```
# reading the first element of corpus
as.character(sms_corpus[[1]])
```

```
## [1] "Hope you are having a good week. Just checking in"
```

3.1.3 To see multiple massages

```
# reading the first three elements of the corpus
lapply(sms_corpus[1:3], as.character)
```

```
## $`1`
## [1] "Hope you are having a good week. Just checking in"
##
## $`2`
## [1] "K..give back my thanks."
##
## $`3`
## [1] "Am also doing in cbe only. But have to pay."
```

3.2 Text Clean-up

3.2.1 All lower case characters

```
# transform all the text to lower case
sms_corpus_clean = tm_map(sms_corpus, content_transformer(tolower))
# Explore whether the change to lower case has been made
as.character(sms_corpus[[1]])
```

```
## [1] "Hope you are having a good week. Just checking in"
```

```
as.character(sms_corpus_clean[[1]])
```

```
## [1] "hope you are having a good week. just checking in"
```

3.2.2 remove numbers

```
# remove all the numbers from the text
sms_corpus_clean = tm_map(sms_corpus_clean, removeNumbers)
# Explore whether the change of removing numbers has been made
as.character(sms_corpus[[4]])
```

```
## [1] "complimentary 4 STAR Ibiza Holiday or £10,000 cash needs your URGENT collection.
```

```
as.character(sms_corpus_clean[[4]])
```

```
## [1] "complimentary  star ibiza holiday or £, cash needs your urgent collection.  now
```

3.2.3 remove the stop words

```
# removing the stop words
sms_corpus_clean = tm_map(sms_corpus_clean, removeWords, stopwords())
# Explore if the stop words has been removed successfully
as.character(sms_corpus[[1]])
```

```
## [1] "Hope you are having a good week. Just checking in"
```

```
as.character(sms_corpus_clean[[1]])
```

```
## [1] "hope     good week. just checking "
```

3.2.4 remove the punctuation

```
# defining a function to remove the punctuation
replacePunctuation = function(x) {gsub("[[:punct:]]+", " ", x)}

# use the replacePunctuation function to remove the punctuation
sms_corpus_clean = tm_map(sms_corpus_clean, replacePunctuation)
# Explore if the punctuation has been removed
as.character(sms_corpus[[2]])
```

```
## [1] "K..give back my thanks."
```

```
as.character(sms_corpus_clean[[2]])
```

```
## [1] "k give back  thanks "
```

3.2.5 stemming (reduce words to roots)

```
# install.packages("SnowballC")
library(SnowballC)
# reform the words to their roots
sms_corpus_clean = tm_map(sms_corpus_clean, stemDocument)
# check whether the stemming has been worked properly
as.character(sms_corpus[[50]])
```

```
## [1] "Yup. Izzit still raining heavily cos i'm in e mrt i can't c outside."
```

```
as.character(sms_corpus_clean[[50]])
```

```
## [1] "yup izzit still rain heavili cos e mrt c outsid"
```

3.2.6 remove additional white space

```
# removing the white spaces
sms_corpus_clean = tm_map(sms_corpus_clean, stripWhitespace)
# Examine whether the white spaces has been removed properly
as.character(sms_corpus[[1]])
```

```
## [1] "Hope you are having a good week. Just checking in"
```

```
as.character(sms_corpus_clean[[1]])
```

```
## [1] "hope good week just check"
```

3.3 Splitting text documents into words (tokenization)

```
# Creating a matrix of words with DocumentTermMatrix
sms_dtm = DocumentTermMatrix(as.factor(unlist(sms_corpus_clean)))
sms_dtm$ncol
```

```
## [1] 6078
```

```
sms_dtm$nrow
```

```
## [1] 5559
```

```
# examples
sms_dtm$dimnames$Terms[1:3]
```

```
## [1] "check" "good"  "hope"
```

```
# exploring sms_dtm
sms_dtm
```

```
## <<DocumentTermMatrix (documents: 5559, terms: 6078)>>
## Non-/sparse entries: 42735/33744867
## Sparsity           : 100%
## Maximal term length: 34
## Weighting          : term frequency (tf)
```

```
str(sms_dtm)
```

```
## List of 6
##  $ i       : int [1:42735] 1 1 1 1 1 2 2 2 3 3 ...
##  $ j       : int [1:42735] 1 2 3 4 5 6 7 8 9 10 ...
##  $ v       : num [1:42735] 1 1 1 1 1 1 1 1 1 1 ...
##  $ nrow    : int 5559
##  $ ncol    : int 6078
##  $ dimnames:List of 2
##   ..$ Docs : chr [1:5559] "1" "2" "3" "4" ...
##   ..$ Terms: chr [1:6078] "check" "good" "hope" "just" ...
##  - attr(*, "class")= chr [1:2] "DocumentTermMatrix" "simple_triplet_matrix"
##  - attr(*, "weighting")= chr [1:2] "term frequency" "tf"
```

4 Creating Training and Testing Datasets

4.1 Creating (75%) training and (25%) testing sets

```
sms_dtm_train = sms_dtm[1:4169,]
sms_dtm_test = sms_dtm[4170:5559,]
```

4.2 Creating lables

```
sms_train_lable = sms_raw[1:4169,]$type
sms_test_lable = sms_raw[4170:5559,]$type
```

4.3 Proportion in training and testing

```
prop.table(table(sms_train_lable))
```

```
## sms_train_lable
##       ham      spam
## 0.8647158 0.1352842
```

```
prop.table(table(sms_test_lable))
```

```
## sms_test_lable
##       ham      spam
## 0.8683453 0.1316547
```

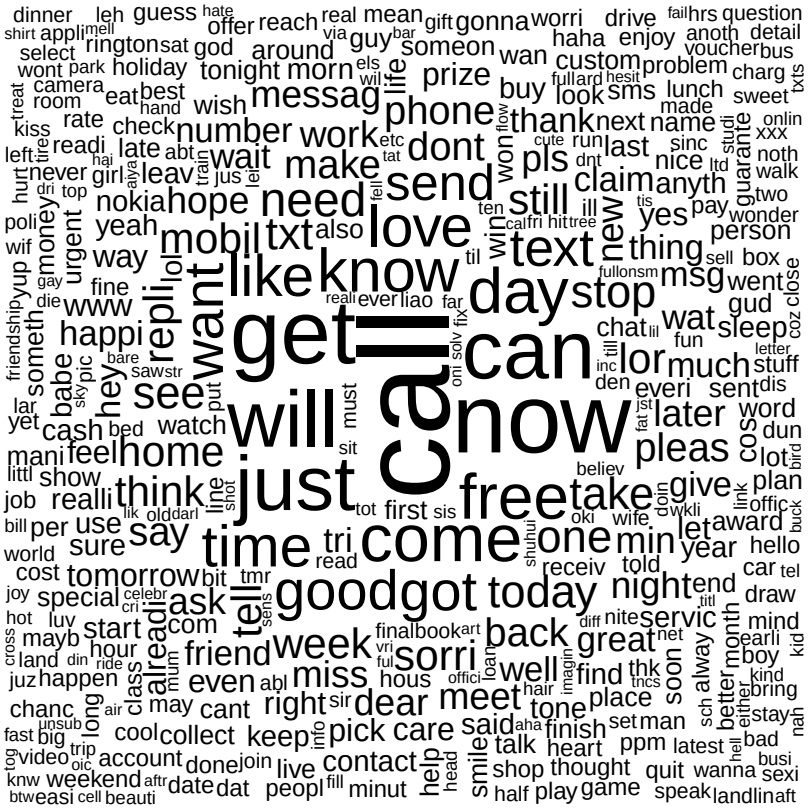
The proportion of two classes ham ans spam in training is similar to test data set.

5 Word cloud - Visualize

```
# install.packages("wordcloud")
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```
wordcloud(unlist(sms_corpus_clean), min.ferq = 50, random.order = FALSE)
```

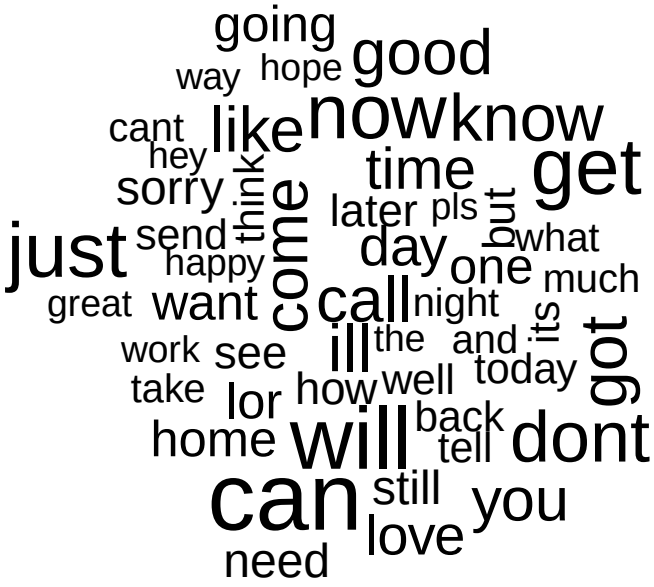


5.1 Visualize cloud for spam

```
spam = subset(sms_raw, type == "spam")
wordcloud(spam$text, max.words = 50, scale = c(3, 0.5))
```

5.2 Visualize cloud for ham

```
ham = subset(sms_raw, type == "ham")
wordcloud(ham$text, max.words = 50, scale = c(3, 0.5))
```



6 Reduce Dimentionality

6.1 Finding the most frequent words

```
sms_freq_words = findFreqTerms(sms_dtm_train, 5)
# The first 10 most frequent words in training set
sms_freq_words[1:10]
```

```
##  [1] "check" "good"  "hope"  "just"  "week"  "back"  "give"  "thank" "also"
## [10] "pay"
```

6.2 Creat DTM with the most frequent words

```
sms_dtm_ferq_train = sms_dtm_train[,sms_freq_words]
sms_dtm_freq_test = sms_dtm_test[,sms_freq_words]
# Comparing the columns before vs after using frequent words
sms_dtm_train$ncol
```

```
## [1] 6078
```

```
sms_dtm_ferq_train$ncol
```

```
## [1] 1167
```

```
convert_counts = function(x) {
  x = ifelse(x > 0, "Yes", "No")
}
```

```
# applying convert count function to reform the train and test sets
sms_train = apply(sms_dtm_ferq_train, MARGIN = 2, convert_counts)
sms_test = apply(sms_dtm_freq_test, MARGIN = 2, convert_counts)
```

7 Train a Model on the train data using Naive bayes algorithm

```
# install.packages("e1071")
library(e1071)
sms_classifier = naiveBayes(sms_train, sms_train_lable)
```

8 Predict and Evaluate the Model Performance

```
sms_test_pred = predict(sms_classifier, sms_test)
```

8.1 Confusion Matrix

```
library(gmodels)

CrossTable(sms_test_pred, sms_test_lable,
           prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,
           dnn = c("actual", "predicted"))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Col Total |
## |-------------------------|
##
##
## Total Observations in Table:  1390
##
##
##              | predicted
##       actual |       ham |      spam | Row Total |
## -------------|-----------|-----------|-----------|
##          ham |      1203 |        22 |      1225 |
##              |     0.997 |     0.120 |           |
## -------------|-----------|-----------|-----------|
##         spam |         4 |       161 |       165 |
##              |     0.003 |     0.880 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      1207 |       183 |      1390 |
##              |     0.868 |     0.132 |           |
## -------------|-----------|-----------|-----------|
##
##
```

The model missclassified 0.12 of ham text as spam and 0.003 of spam as ham. It seems a good model in overal but we can check the Naive Bayes considering Laplace method to see how it will change.

9 Improving the Model Performance by applyiong Laplace

```
sms_classifier2 = naiveBayes(sms_train, sms_train_lable, laplace = 1)
sms_test_pred2 = predict(sms_classifier2, sms_test)


CrossTable(sms_test_pred2, sms_test_lable,
           prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,
           dnn = c("actual", "predicted"))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Col Total |
## |-------------------------|
##
##
## Total Observations in Table:  1390
##
##
##              | predicted
##       actual |       ham |      spam | Row Total |
## -------------|-----------|-----------|-----------|
##          ham |      1190 |        10 |      1200 |
##              |     0.986 |     0.055 |           |
## -------------|-----------|-----------|-----------|
##         spam |        17 |       173 |       190 |
##              |     0.014 |     0.945 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      1207 |       183 |      1390 |
##              |     0.868 |     0.132 |           |
## -------------|-----------|-----------|-----------|
##
##
```

```
print("The model without laplace")
```

```
## [1] "The model without laplace"
```

```
table(sms_test_pred, sms_test_lable)
```

```
##              sms_test_lable
## sms_test_pred  ham spam
##          ham  1203   22
##          spam    4  161
```

```
print("The model with laplace")
```

```
## [1] "The model with laplace"
```

```
table(sms_test_pred2, sms_test_lable)
```

```
##               sms_test_lable
## sms_test_pred2  ham spam
##           ham  1190   10
##           spam   17  173
```

As we can see in the table above, and comparing two confusion matrices, Model 1 without using Laplace method is preferable, since the missed classified labels for spam as ham is increased in the second model.

THE END