Spam

Thadryan Sweeney

November 7, 2017

```
# read and inspect the raw data
sms_raw <- read.csv('C:/Users/Thadryan.Hank-PC/Documents/R/da5030.spammsgdataset.csv', stringsAsFactors
str(sms_raw)
```

```
## 'data.frame':    5574 obs. of  2 variables:
##  $ type: Factor w/ 2 levels "ham","spam": 1 1 2 1 1 2 1 1 2 2 ...
##  $ text: Factor w/ 5160 levels "'An Amazing Quote'' - \"Sometimes in life its difficult to decide wha
```

We'll make a table to get an idea of the data looks like in terms of our areas of interest, in this case, spam and ham. We see the the majority of the set is composed of what we're looking for, but the unwanted messages are common enough to be problematic.

```
# create a table based on type
table(sms_raw$type)
```

```
##
##  ham spam
## 4827  747
```

R has great library support in general, and text mining is no exception. We can import the tm library and use it to create a corpus of data

```
# this is a library for text mining
library(tm)
```

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

```
# we make a body of text to mine. The "V" stands for volatile, meaning it is not store permanantly on t
sms_corpus <- VCorpus(VectorSource(sms_raw$text))

# display the traits
print(sms_corpus)
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 5574
```

```
# inspec the first two
inspect(sms_corpus[1:2])
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 2
##
## [[1]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 111
##
## [[2]]
## <<PlainTextDocument>>
```

```
## Metadata:  7
## Content:  chars: 29
```

```r
# view the content of a message
as.character(sms_corpus[[1]])
```

```
## [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there g
```

```r
# view more than one
lapply(sms_corpus[1:2], as.character)
```

```
## $`1`
## [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there g
##
## $`2`
## [1] "Ok lar... Joking wif u oni..."
```

```r
# start cleaning text
sms_corpus_clean <- tm_map(sms_corpus, content_transformer(tolower))

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

```
## [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there g
```

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

```
## [1] "go until jurong point, crazy.. available only in bugis n great world la e buffet... cine there g
```

```r
# update the data with removal of numbers, stopwords (to, but, and), and punctuation
sms_corpus_clean <- tm_map(sms_corpus_clean, removeNumbers)
sms_corpus_clean <- tm_map(sms_corpus_clean, removeWords, stopwords())
sms_corpus_clean <- tm_map(sms_corpus_clean, removePunctuation)
```

Next we will convert works into thier roots, for example "writing" becomes "write". This keeps the themes legitimate without cluttering the dataset with redundancy and making it difficult to count.

```r
# this library will allow use to convert forms of the words to roots
library(SnowballC)
```

Numerous steps still need to take place before the dataset is ready to use, however. We'll need to remove whitespace and create a matrix. There is also a function that could speed up this process for next time now that we understand it.

```r
sms_corpus_clean <- tm_map(sms_corpus_clean, stemDocument)

# remove extra whitespace
sms_corpus_clean <- tm_map(sms_corpus_clean, stripWhitespace)

# create a document term matrix from the data
sms_dtm <- DocumentTermMatrix(sms_corpus_clean)

#demostrate function parameters that could speed up the whole process
sms_dtm2 <- DocumentTermMatrix(sms_corpus, control = list(
  tolower = TRUE,
  removeNumbers = TRUE,
  stopwords = TRUE,
  removePunctuation = TRUE,
  stemming = TRUE))
sms_dtm
```

```
## <<DocumentTermMatrix (documents: 5574, terms: 6604)>>
## Non-/sparse entries: 42631/36768065
## Sparsity           : 100%
## Maximal term length: 40
## Weighting          : term frequency (tf)
```

```
sms_dtm2
```

```
## <<DocumentTermMatrix (documents: 5574, terms: 6998)>>
## Non-/sparse entries: 43720/38963132
## Sparsity           : 100%
## Maximal term length: 40
## Weighting          : term frequency (tf)
```

Partition the data

Now we can create training and validation datasets, and create the labels for what we are trying to predict. We will also observe the proportions in the datasets, to ensure we haven't accidentally loaded the dice by putting a drastically different proportion in one than the other.

```
# break into testing and training set
sms_dtm_train <- sms_dtm[1:4169, ]
sms_dtm_test <- sms_dtm[4170:5559, ]


# set labels while we are at it
sms_train_labels <- sms_raw[1:4169, ]$type
sms_test_labels <- sms_raw[4170:5559, ]$type

# confirm we are on the right track
prop.table(table(sms_train_labels))
```

```
## sms_train_labels
##       ham      spam
## 0.8647158 0.1352842
```

```
prop.table(table(sms_test_labels))
```

```
## sms_test_labels
##       ham      spam
## 0.8697842 0.1302158
```

Visualize the data

Now that we have that taken care of, we can proceede though some analytical steps and then build our model. One thing we want to know about is word frequency. Wordclouds are a great way to get an idea of paterns intuitively.

```
# get the library
library(wordcloud)
```

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

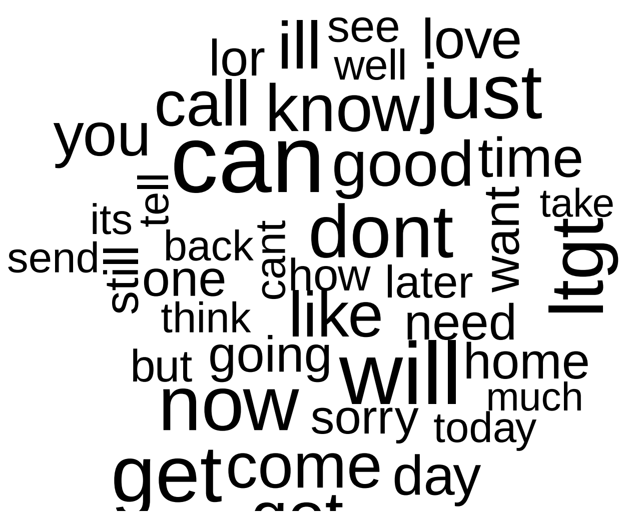
```r
#call the function
wordcloud(sms_corpus_clean, min.freq = 50, random.order = FALSE)
```

Let's compare that to the junk folder. If we're on the right track, there should be a difference, which we will later quantify and use for our classifier.

```r
# call the subsets
spam <- subset(sms_raw, type == "spam")
ham <- subset(sms_raw, type == "ham")

# pass to the wordcloud function
wordcloud(spam$text, max.words = 40, scale = c(3, 0.5))
```


```r
wordcloud(ham$text, max.words = 40, scale = c(3, 0.5))
```



Now let's get more quantitative about the frequency, finding some areas the keep popping up.

```
# call the frequency counter with argument for number of occurences
sms_freq_words <- findFreqTerms(sms_dtm_train, 5)
str(sms_freq_words)
```

```
##  chr [1:1158] "â<U+0080><U+0093>" "abiola" "abl" "abt" "accept" "access" "account" ...
```

```
# see if there is a difference in the sets
sms_dtm_freq_train <- sms_dtm_train[ , sms_freq_words]
sms_dtm_freq_test <- sms_dtm_test[ , sms_freq_words]
```

We will now make a function to convert to yes/no values for a more reader friendly output.

```
# define function
convert_counts <- function(x)
{
  x <- ifelse(x > 0, "yes", "no")
}

# apply the function by rows to the datasets
sms_train <- apply(sms_dtm_freq_train, MARGIN = 2, convert_counts)
sms_test <- apply(sms_dtm_freq_test, MARGIN = 2, convert_counts)
```

Next we will get a few more libraries. These will be used for the actual classification and to see how our model preforms.

```
# get required materials
library("e1071")
```

```r
library("gmodels")

# call the naive bayes function
sms_classifier <- naiveBayes(sms_train, sms_train_labels)

# use it to make a predition
sms_test_pred <- predict(sms_classifier, sms_test)

# inspect the results
CrossTable(sms_test_pred, sms_test_labels, prop.chisq = FALSE, prop.t = FALSE, dnn = c('predicted', 'ac
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |-------------------------|
##
##
## Total Observations in Table:  1390
##
##
##              | actual
##    predicted |       ham |      spam | Row Total |
## -------------|-----------|-----------|-----------|
##          ham |      1200 |        20 |      1220 |
##              |     0.984 |     0.016 |     0.878 |
##              |     0.993 |     0.110 |           |
## -------------|-----------|-----------|-----------|
##         spam |         9 |       161 |       170 |
##              |     0.053 |     0.947 |     0.122 |
##              |     0.007 |     0.890 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      1209 |       181 |      1390 |
##              |     0.870 |     0.130 |           |
## -------------|-----------|-----------|-----------|
##
##
```

Ont thing we haven't talked about so far in this context is the Laplace approximator. Naive Bayes classifiers work by multiplying probabiliteis derived from imperical values. This means that if we're looking at a term that didn't occur, we end up multiplying by zero and nullifying our results. To avoid having the function need to work around this, we can simply replace the value with a 1, which will likely cause very little, if any, perceptible disturbance in the functions.

```r
# add laplace argument, replacing zeros with ones
sms_classifier2 <- naiveBayes(sms_train, sms_train_labels, laplace = 1)

# make new predictor
sms_test_pred2 <- predict(sms_classifier2, sms_test)

# visualize the results
CrossTable(sms_test_pred2, sms_test_labels, prop.chisq = FALSE, prop.t = FALSE, dnn = c('predicted', 'a
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |-------------------------|
##
##
## Total Observations in Table:  1390
##
##
##              | actual
##    predicted |       ham |      spam | Row Total |
## -------------|-----------|-----------|-----------|
##          ham |      1202 |        28 |      1230 |
##              |     0.977 |     0.023 |     0.885 |
##              |     0.994 |     0.155 |           |
## -------------|-----------|-----------|-----------|
##         spam |         7 |       153 |       160 |
##              |     0.044 |     0.956 |     0.115 |
##              |     0.006 |     0.845 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      1209 |       181 |      1390 |
##              |     0.870 |     0.130 |           |
## -------------|-----------|-----------|-----------|
##
##
```