Spam Detection — Naive Bayes Classifier

Sean O'Malley

5/29/2017

Objective : Classify SMS message as spam or not spam (ham).

Use Bayes theorem (or its extension) and write down the corresponding conditional probability of the study problem

$P(A|B) = (P(A) * P(A|B))/P(B)$

$P(spam|ham) = (P(spam) * P(spam|ham))/P(ham)$

```
sms <- read.delim("/Users/SeanOMalley1/Desktop/MSDS\ 680\ ML/smsspamcollection/SMSSpamCollection",
                  quote = "",
                  sep = "\t",
                  header = F,
                  col.names = c("type", "message"),
                  stringsAsFactors = F)

sms$type <- as.factor(sms$type)
```

Quick Data Exploration

```
glimpse(sms)
```

```
## Observations: 5,574
## Variables: 2
## $ type    <fctr> ham, ham, spam, ham, ham, spam, ham, ham, spam, spam,...
## $ message <chr> "Go until jurong point, crazy.. Available only in bugi...
```

```
summary(sms$type)
```

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

Use Naïve Bayes to classify the SMS message.

Framework for text classification:

- Transformation (change to lower case, remove numbers, remove punctuation, stop words, white space, word stemming, etc.)
- Document-Term-Matrix creation – matrix of word counts for each individual document in the matrix (e.g. documents as rows, words as columns or vice versa)
- Text Analysis (e.g. word counts, visualizations using wordclouds)

Clean and Standardize Text

We must first create a corpus, which is a collection of text documents. Doing so we see that we have created 5574 sms messages for the training data.

```
sms_corpus <- VCorpus(VectorSource(sms$message))

print(sms_corpus)
```

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

We can now use list operators to search documents within sms_corpus. The below command offers a summary of those documents.

```
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
```

To view the actual messsage, use as.character function on the list subset

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

```
## [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine t
here got amore wat..."
```

Use lapply of as.character onto sms_corpus to view several messages at once

```
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 t
here got amore wat..."
##
## $`2`
## [1] "Ok lar... Joking wif u oni..."
```

Okay, enough playing around here, in order to perform our analysis, we need to divide these messages into individual words. We will apply text mining to our sms_corpus list with the tm_map function. We will also normalize the data further with the tolower function

```
sms_corpus_clean <- tm_map(sms_corpus, content_transformer(tolower))
```

Lets inspect this function to see if it worked. Cool, everything looks lower cased.

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

```
## [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine t
here got amore wat..."
```

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

```
## [1] "go until jurong point, crazy.. available only in bugis n great world la e buffet... cine t
here got amore wat..."
```

Now lets get rid of numbers, because for the question we are asking numbers are relavant to the message but not necessarily useful when considering spam vs. non-spam,

```
sms_corpus_clean <- tm_map(sms_corpus_clean, removeNumbers)
```

Lets inspect this function to see if it worked. Sweet, everything looks like we got rid of numerics.

```
as.character(sms_corpus[[10]])
```

```
## [1] "Had your mobile 11 months or more? U R entitled to Update to the latest colour mobiles wit
h camera for Free! Call The Mobile Update Co FREE on 08002986030"
```

```
as.character(sms_corpus_clean[[10]])
```

```
## [1] "had your mobile  months or more? u r entitled to update to the latest colour mobiles with
 camera for free! call the mobile update co free on "
```

Now its time to remove filler or stop words. We can continue to do so using the tm_map function. The removeWords argument tells us to remove words and stopwords argument gives us a predefined list of words to look for that meet our specifications to remove.

```
sms_corpus_clean <- tm_map(sms_corpus_clean, removeWords, stopwords())
```

Lets inspect this function to see if it worked. Great, everything looks like we got rid of stop words.

```
as.character(sms_corpus[[6]])
```

```
## [1] "FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you u
p for it still? Tb ok! XxX std chgs to send, £1.50 to rcv"
```

```
as.character(sms_corpus_clean[[6]])
```

```
## [1] "freemsg hey  darling    week's now   word back!  like  fun     still? tb ok! xxx std chgs
send, £.  rcv"
```

Now lets eliminate punctuation.

```
sms_corpus_clean <- tm_map(sms_corpus_clean, removePunctuation)
```

Looks like it worked.

```
as.character(sms_corpus[[3]])
```

```
## [1] "Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to re
ceive entry question(std txt rate)T&C's apply 08452810075over18's"
```

```
as.character(sms_corpus_clean[[3]])
```

```
## [1] "free entry    wkly comp  win fa cup final tkts st may  text fa    receive entry questionst
d txt ratetcs apply s"
```

We will now perform stemming, which reduces all words down to their root word. We will do so using the wordStem function within the SnowballC package.

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

Lets see if it worked.

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

```
## [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine t
here got amore wat..."
```

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

```
## [1] "go jurong point crazi avail bugi n great world la e buffet cine got amor wat"
```

Lastly, with all the changes we've made, we have left q quite a bit of white space, lets get rid of it.

```
sms_corpus_clean <- tm_map(sms_corpus_clean, stripWhitespace)
```

Looks good.

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

```
## [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine t
here got amore wat..."
```

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

```
## [1] "go jurong point crazi avail bugi n great world la e buffet cine got amor wat"
```

Splitting Documents into Words

This process is called Tokenization, and a token is defined as a single element of a text string. We will do so using the DocumentTermMatrix function, which takes a corpus and creates a data structure called a DTM. DTM rows indicate documents (SMS messages) and columns indicate terms (words).

Each cell in the matrix stores a number indicating a count of the times the word represented by the column appears in the document represented by the row.

```
sms_dtm <- DocumentTermMatrix(sms_corpus_clean)
```

Lets have a look at the sparse matrix we've created. Lookin good!

```
sms_dtm
```

```
## <<DocumentTermMatrix (documents: 5574, terms: 6577)>>
## Non-/sparse entries: 42605/36617593
## Sparsity           : 100%
## Maximal term length: 40
## Weighting          : term frequency (tf)
```

Establish Test and Training Datasets

```
# split out the return of test and train into their own data frames
sms_dtm_train <- sms_dtm[1:4169,]

sms_dtm_test <- sms_dtm[4170:5574,]

# Save pair of vectors with labels for each of the rows in the training and testing matricies
sms_train_labels <- sms[1:4169,]$type

sms_test_labels <- sms[4170:5574,]$type

# confirm subsets are representative of the complete set of SMS data
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
## 0.8697509 0.1302491
```

Visualize Text Data

A word cloud is a way to visually depict the frequency at which words appear in text data.

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

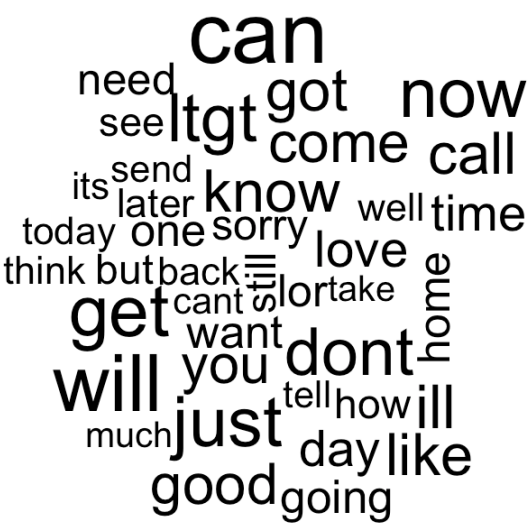
A more interesting visualization involves comparing the clouds for SMS spam and ham. Given a vector of raw text strings, it will automatically apply common text preparation processes before displaying the cloud.

Lets get a subset of sms data by type first

```
spam <- subset(sms, type == "spam")

ham <- subset(sms, type == "ham")
```

We now have two data frames, each with a text feature containing the raw text strings for SMS's. We can now create two different word clouds specific to whether an email is spam or ham.

```
par(mfrow=c(1,2))

# spam
wordcloud(spam$message, max.words = 40, scale = c(3, 0.5))

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



The above differences in word clouds suggest that our Naive Bayes model will have some strong key words to differentiate between the classes.

Create Indicator Features for Frequent Words

We need to now transform our sparse matrix into a data structure that can be used for our Naive Bayes classifier. To reduce the number of features further, we will eliminate any word that appear in less than five SMS messages, or in less than 0.1% of records.

The below command displays the words appearing at least 5 ties in the train dataset.

```
sms_freq_words <- findFreqTerms(sms_dtm_train, 5)

str(sms_freq_words)
```

```
##  chr [1:1157] "abiola" "abl" "abt" "accept" "access" ...
```

We now need to filter our DTM to include only the terms appearing in a specified vector. The below command only shows the columns representing the words in the sms_freq_words vector for each the training and testing dataset.

```
sms_dtm_freq_train <- sms_dtm_train[ , sms_freq_words]

sms_dtm_freq_test <- sms_dtm_test[ , sms_freq_words]
```

The Naive Bayes classifier is typically trained on data with categorical features. We need to change this to a categorical variable that simply indicated yes or no depending on whether the word appears at all. We then will apply this to the dtm train / test datasets.

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

sms_train <- apply(sms_dtm_freq_train, MARGIN = 2, convert_counts)

sms_test <- apply(sms_dtm_freq_test, MARGIN = 2, convert_counts)
```

The result is in two character type matrices, each, with cells indicating “Yes” or “No” for whether the word represented by the column appears at any point in the message represented by the row.

Train Model on Data

We will use the e1071 package in employ the Naive Bayes functionality.

We will build a classifier:

```
sms_classifier <- naiveBayes(sms_train, sms_train_labels)
```

Test Prediction on Test Data

We need to test the classifier’s predictions on unseen messages in the test data. We will use the predict funtion to determine how our classifier performs against the new test data.

```
sms_test_pred <- predict(sms_classifier, sms_test)
```

We will use a cross table to compare predictions to the true values.

```
CrossTable(sms_test_pred, sms_test_labels,
           prop.chisq = F, prop.t = F,
           dnn = c('predicted','actual'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |-------------------------|
##
##
## Total Observations in Table:  1405
##
##
##              | actual
##    predicted |       ham |      spam | Row Total |
## -------------|-----------|-----------|-----------|
##          ham |      1213 |        20 |      1233 |
##              |     0.984 |     0.016 |     0.878 |
##              |     0.993 |     0.109 |           |
## -------------|-----------|-----------|-----------|
##         spam |         9 |       163 |       172 |
##              |     0.052 |     0.948 |     0.122 |
##              |     0.007 |     0.891 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      1222 |       183 |      1405 |
##              |     0.870 |     0.130 |           |
## -------------|-----------|-----------|-----------|
##
##
```

Initial Results Interpretation

We only misqualified 29 of 1405 spam v. ham in our predictions, which is not too bad. This exemplifies the reason why Naive Bayes is the standard for text classification. Nevertheless, lets see if we can try and improve the model a bit.

Improve Model Performance

Lets add a laplace estimator to allow words that appeared in zero spam or zero ham messages to have an indisputable say in the classification process.

```
# build classifier2
sms_classifier2 <- naiveBayes(sms_train, sms_train_labels, laplace = 1)

# predict using classifier2
sms_test_pred2 <- predict(sms_classifier2, sms_test)

# test with crosstable confusion matrix
CrossTable(sms_test_pred2, sms_test_labels,
           prop.chisq = F, prop.t = F, prop.r = F,
           dnn = c('predicted', 'actual'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Col Total |
## |-------------------------|
##
##
## Total Observations in Table:  1405
##
##
##              | actual
##    predicted |       ham |      spam | Row Total |
## -------------|-----------|-----------|-----------|
##          ham |      1215 |        28 |      1243 |
##              |     0.994 |     0.153 |           |
## -------------|-----------|-----------|-----------|
##         spam |         7 |       155 |       162 |
##              |     0.006 |     0.847 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      1222 |       183 |      1405 |
##              |     0.870 |     0.130 |           |
## -------------|-----------|-----------|-----------|
##
##
```

It looks like adding the laplace estimator reduced the number of false positives by 2, but actually increased the number of false negatives by 8. This is not as good as our last model, but makes sense that giving ulimate weight to some words than be good heuristics algorithmically, but can allow non-optimal results when a model is already really accurate.

Conclusion

Naive Bayes constructs tables of probabilities that are used to estimate the likelihood that new examples belong to various classes. The probabilities are calculated using Bayes theorum, which specifies how dependent events are related. Naive assumptions allow us to handle larger datasets in regards to computational cost.

What is the accuracy of the model?

We properly classified 97.935% of emails as spam or ham.

5 most frequent words in each class

spam : call, free, mobile, now, text

ham : will, just, can, get, like

How would you improve the model performance?

I would continue to manipulate the way in which we parse the text data. The methods we used worked okay, but could be more optimal. I think it could be interesting to run some sentiment or other string analysis on prior to putting through the model. Tokenization works great, but context is often lost, so if there is a way for us to have factorized strings or at least result metrics of those combinations, we could increase the accuracy of the model.

If the data set is bigger, do you think the accuracy increases? Discuss.

We can assume with machine learning, the more relavant and contextually appropriate data available, the better. Every single time we can have a larger dataset, the more the model has to both learn and test against to improve performance.