Classifying SMS Messages

Scott Schirkofsky

MSDS 680 Machine Learning

Regis University

Classifying SMS Messages

**From the given data set, use Naïve Bayes to classify the SMS message.**

First the appropriate libraries are loaded.

## load libraries

library(tm) #text mining package: tm\_map()

library(SnowballC) #used for stemming, wordStem(), stemDocument()

library(wordcloud) #wordcloud generator

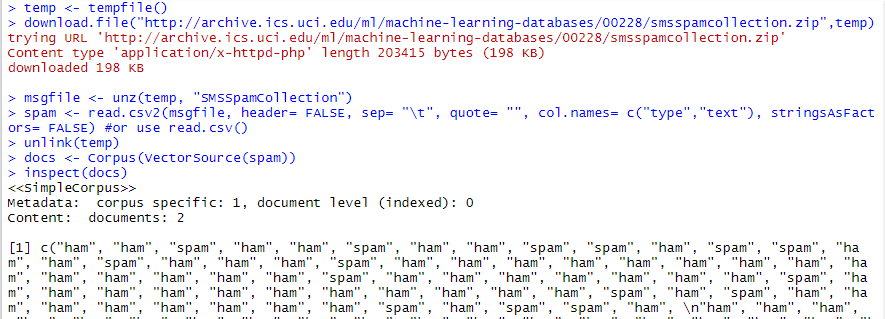
library(e1071) #Naive Bayes

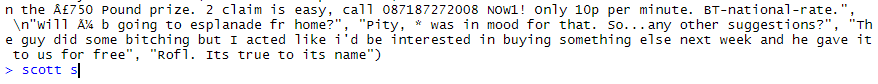
library(gmodels) #CrossTable()

library(caret) #ConfusionMatrix()

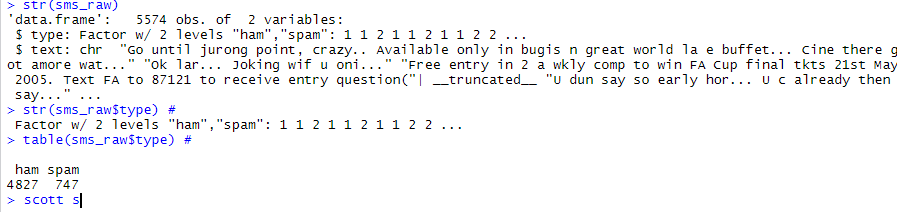
library("RColorBrewer") #color palettes

The data set is read and assigned as a variable and displayed with the string function.





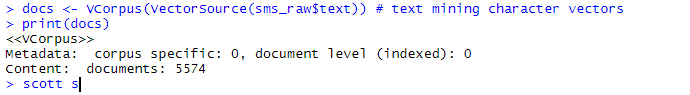
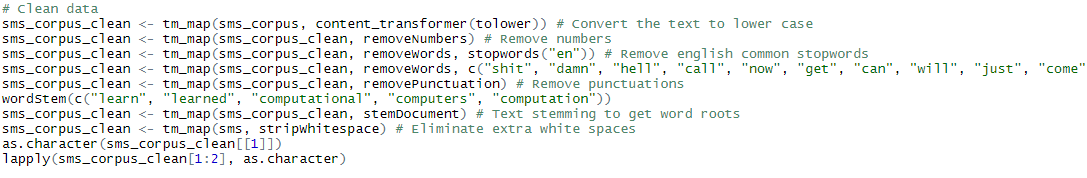
Using the str() function I see that the data frame contains 5,574 objects with two features. “Spam” and “ham” have assigned to the SMS type. The table() functions show that there are 747 messages that are classified as spam.



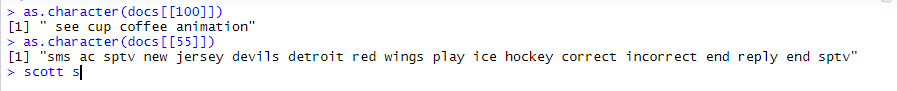
**The framework for text classification is briefly summarized here:**

* 1. **transformation (change to lower case, remove numbers, remove punctuation, stop words, white space, word stemming, etc.)**

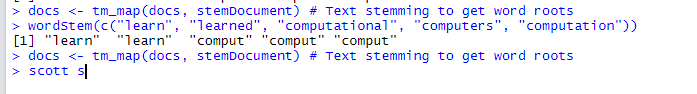
Here I used the tm\_map library to clean up the data. With this, I added additional words to the remove list and added various word to the stemDocument list.



As.character() is run to verify tm\_map "tolower" worked in converting that characters to lower case.

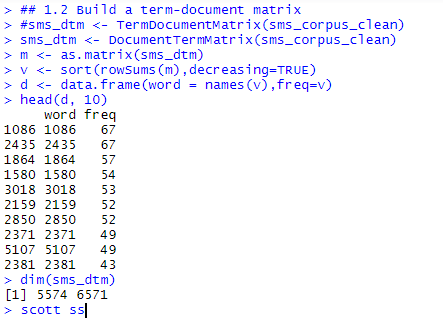


The wordstem() function from the SnowballC library is used to apply stemming on a handful of words.



* 1. **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)**

The TermDocumentMatrix() is used to create a DTM data structure out of the corpus created earlier with Vcorpus(). This process will create a DTM object and by turning the rows into terms and the columns into documents.

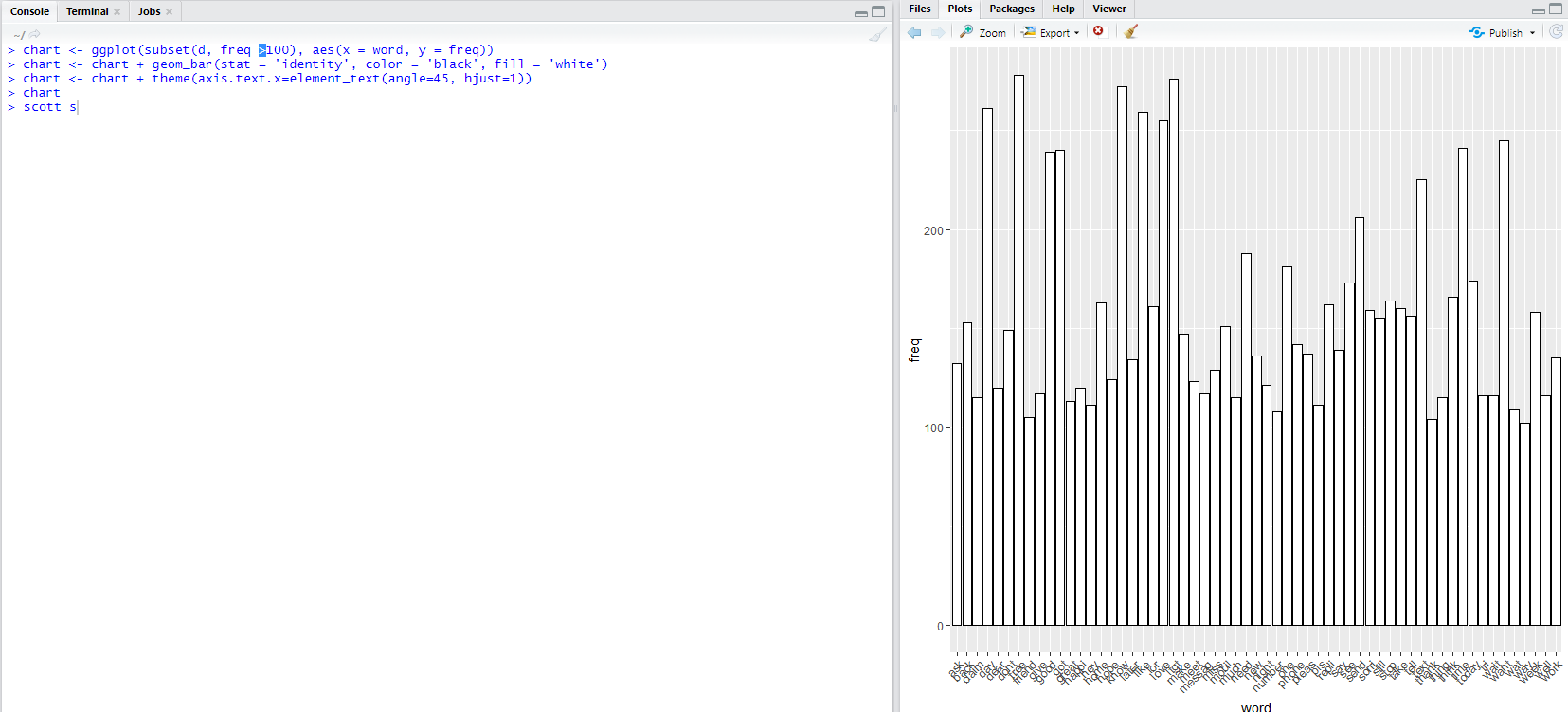


**1.3) Text Analysis (e.g. word counts, visualizations using wordclouds)**

A wordcloud is created from the cleaned data.



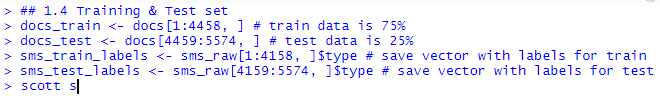
Next the data is plotted using ggplot.



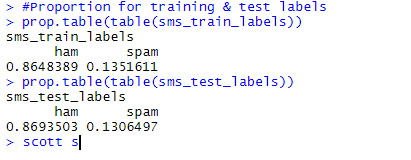
**Answer these questions:**

* 1. **What is the accuracy of the model? Report your finding with corresponding tables/graphs**

The data is first divided 75% / 25% into training and test data sets. A pair of vectors is saved with labels.



Next prop is used to confirm the subsets are a good representation of the SMS data. Both sets contain a little over 13% spam messages, this tells me that the data was evenly divided between train and test with a difference of .51% or half a percent (13.52% – 13.01%).



I ran CrossTable() to test the model by comparing the predictions to the real values. The classifications of interest are the of the “true” variety TP and TN. 129 true positives were predicted along with 962 true negatives (mistakes). This means that there were 129 true positives for heart disease and 962 true negatives or correctly rejected. Using the confusion matrix data, the error rate and accuracy are calculated below. There were 9 FP false alarms (type 1 error), and 16 FN misses (type 2 error).

The model performed very well with a low error rate of 2.2% and 97.8% accuracy.

**Error rate:** (FP + FN) / (TP + TN + FP + FN)

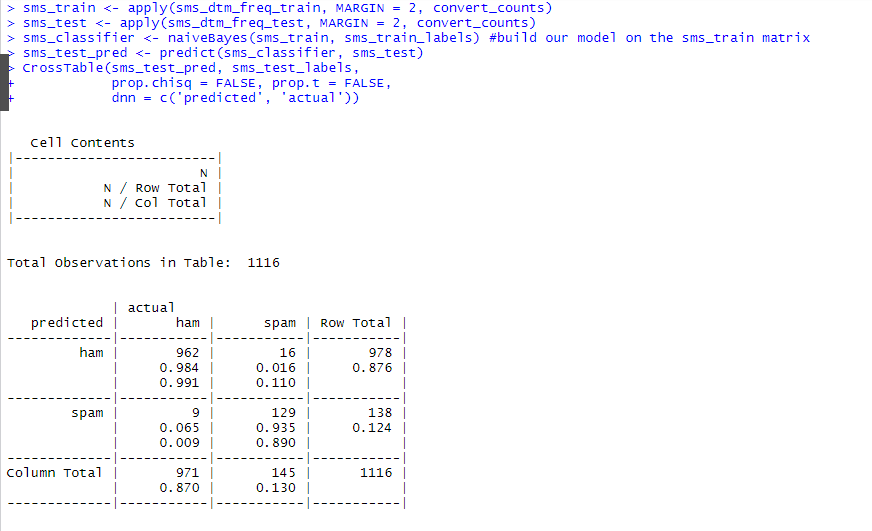
> (9 + 16) / (129 + 962 + 9 + 16)  
 > 25 / 1116  
 > .0224

**Accuracy:** (TP + TN) / (TP + TN + FP + FN)  
 > (129 + 962) / (129 + 962 + 9 + 16)  
 > 1091 / 1116  
 > .9776

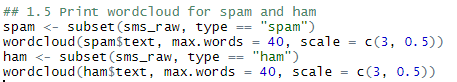
**True Positive:** 129  
**True Negative:** 962

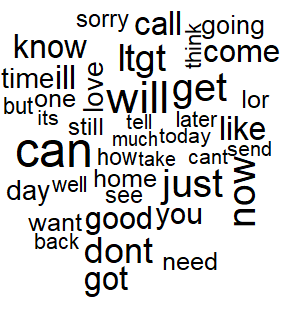
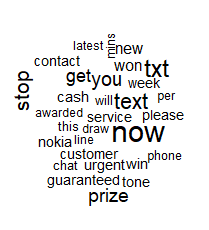
**False Positive:** 9 (Type 1 error)

**False Negative:** 16 (Type 2 error)



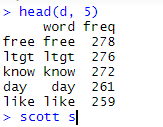
* 1. **Create word clouds for both ham and spam data sets (make sure the plots are readable, you may need to adjust some parameters)**





* 1. **Print the 5 most frequent words (in order from highest to lowest) for each class (both ham class and spam class)**

First I print the overall top 5 frequent used terms using the head() function.

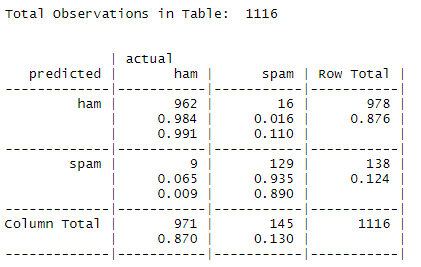


Next, I want to print the top 5 for each of the ham and spam subsets but cannot figure out how to do so. I believe that I’m supposed to use them in a manner similar to wordcloud where I reference “spam$text” along with a way to limit the count to top N words similar to head(x, 5). However, I’m not sure how to accomplish this. I’ve also experimented with finFreqTerms() with no luck. I’ve left some of my attempted code in my r document in section 1.6.

* 1. **Conclude your findings (discussion/insights on the experiments)**

Overall, the model performed very well with a low error rate of 2.2% and 97.8% accuracy. The probabilities were calculated using the naiveBayes() classifier function. Because the model is considered a success as it was able to classify almost 98% of all the SMS messages correctly as “spam” or “ham”.

The table shows me that there were 25 (9 + 16) of the 1,116 messages were incorrectly classified. That’s 2.2%. 9 of the messages were falsely identified as spam, and 16 of the 145 spam messages were incorrectly labeled as ham.



The classifications of interest are the of the “true” variety TP and TN. 129 true positives were predicted along with 962 true negatives (mistakes). This means that there were 129 true positives for heart disease and 962 true negatives or correctly rejected. Using the confusion matrix data, the error rate and accuracy are calculated below. There were 9 FP false alarms (type 1 error), and 16 FN misses (type 2 error).

**Error rate:** (FP + FN) / (TP + TN + FP + FN)

> (9 + 16) / (129 + 962 + 9 + 16)  
 > 25 / 1116  
 > .0224

**Accuracy:** (TP + TN) / (TP + TN + FP + FN)  
 > (129 + 962) / (129 + 962 + 9 + 16)  
 > 1091 / 1116  
 > .9776

**True Positive:** 129  
**True Negative:** 962

**False Positive:** 9 (Type 1 error)

**False Negative:** 16 (Type 2 error)

**1.8) (Extra points) Perform your choices of analysis (not specified above) on the given data set - clearly state your objectives, illustrate the results, and interpret them**

Here I display a confusionMatrix() from the cart library to better display the models accuracy. This allows me to compare predictions from the actual class quickly.

The matrix also shows a sensitivity value of 89%. This represents the number of correct positives. The specificity value, or number of correct negatives is 98%. These numbers tell me that the classifier is correctly predicting the class of the items.

The matrix shows that high positive predictive values and negative predictive values of 93% and 98% indicate that the probability of a true positive/true negative are accurate.

With an overall accuracy of 98% and a p-value of .2301, we can see that our classifier is doing a good job of classifying the sms messages.

