NAIVE BAYES

To illustrate the Naive Bayes method, we'll return to a previous modeling problem which focused on trying to separate legitimate text messages from spam. The data for this example has already been parsed into a term-document matrix (sms_tdm) for the purposes of modeling, but the raw data (sms_raw) is also available on the same .Rdata file. We'll first want to split our data for training and testing. We'll randomly permute the rows of the matrix and then take the first 75% for training. The target variable (with levels of 'ham' or 'spam') is contained in the data frame of raw text messages.

```
> library(tm)
> load('SMS_tdm.Rdata')
> set.seed(11117)
> train=sample(c(T,F),5574, replace=T, p=c(0.75,0.25))
> sms_train = sms_tdm[train,]
> sms_test = sms_tdm[!train,]
> sms_train_labels = sms_raw$type[train]
> sms_test_labels = sms_raw$type[!train]
> # Check that the number of events (spam) sampled in training and test data are comparable
> prop.table(table(sms_train_labels))
```

```
sms_train_labels
      ham      spam
0.8665391 0.1334609
```

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

```
sms_test_labels
      ham      spam
0.8643216 0.1356784
```

We will cut down the size of our dictionary by removing words that do not appear in a given proportion of our documents. We'll start with removing words that appear in <0.1 percent of the corpus (about 5 documents). We'll likely see that increasing this cutoff improves our prediction results. The findFreqTerms() function is contained in the tm package and finds words appearing in more than a given number of documents. We'll use the training set to play with this cutoff:

```
> freqWords = findFreqTerms(sms_train, 5)
> str(freqWords)
```

```
chr [1:1255] "abiola" "abl" "about" "abt" "accept" "access" "account" ...
```

```
> #' This reduces our dictionary size from 6685 to 1160, a huge reduction.
> #' Now let's subset the training/testing data to those words only:
> sms_train = sms_train[ ,freqWords]
> sms_test = sms_test[ ,freqWords]
```

Let's try to use Naive Bayes to classify the text messages. Naive bayes is typically trained on data with categorical inputs. If numeric inputs are used, then a normal distribution is assumed for those variables. In this problem, we want to think of our observations not as a collection of word frequencies (which will certainly not be normally distributed anyway), but merely a collection of words, regardless of how many times the word appeared in the text message. For the vast majority of cases, the word frequency will be 1 anyway. To do that, we will convert our columns to factors with two levels indicating the presence or absense of a word. First we write a function that does this for one column, then we apply it to all columns: In the apply function, margin=2 says to operate along columns as opposed to rows (margin=1).

```
> convert_counts <- function(x)
+ {y = ifelse(x>0,"yes","no")
+  return(y) }
> sms_train1 = apply(sms_train,2,convert_counts)
> sms_test1 = apply(sms_test,2,convert_counts)
```

Then we can train the model on our training data and see how it performs on the test dataset:

```
> library(e1071) # Where the naiveBayes function is found
> library(gmodels) # Where the CrossTable function is found
> model = naiveBayes(sms_train1, sms_train_labels)
> pred = predict(model,sms_test1, type="class")
> CrossTable(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:  1393


             | Actual
   Predicted |       ham |      spam | Row Total |
-------------|-----------|-----------|-----------|
         ham |      1201 |        16 |      1217 |
             |     0.987 |     0.013 |     0.874 |
             |     0.998 |     0.085 |           |
-------------|-----------|-----------|-----------|
        spam |         3 |       173 |       176 |
             |     0.017 |     0.983 |     0.126 |
             |     0.002 |     0.915 |           |
-------------|-----------|-----------|-----------|
Column Total |      1204 |       189 |      1393 |
             |     0.864 |     0.136 |           |
-------------|-----------|-----------|-----------|
```

```
>
```

Can we improve model performance? Having a high false positive rate is undesirable because it could result in someone missing an important text because we thought it was spam. The Laplace estimator is one dial that will help tune the model. When the Laplace estimator is 0, words that did not appear in any spam messages has an indisputable influence on the classification - because $P(x_k|c_i) = 0 \Rightarrow P(\mathbf{x}|c_i) = 0$ (even if all other probabilities are large!) but simply because the word 'hello' may have appeared only in ham messages from our sample, it does not mean that every message containing that word should be classified as ham.

The Laplace correction (or Laplace estimator) essentially adds a small specified number to every cell count in our table of words by class. If the sample size is sufficient, this will do very little to alter the predicted probabilities, except when the predicted probability is 0 (which is undesirable). Let's see how we do when we add in this correction:

```
> model2 = naiveBayes(sms_train1, sms_train_labels, laplace = 1)
> pred2 = predict(model2,sms_test1, type="class")
> CrossTable(pred2, 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:  1393


             | Actual
   Predicted |       ham |      spam | Row Total |
-------------|-----------|-----------|-----------|
         ham |      1195 |        11 |      1206 |
             |     0.991 |     0.009 |     0.866 |
             |     0.993 |     0.058 |           |
-------------|-----------|-----------|-----------|
        spam |         9 |       178 |       187 |
             |     0.048 |     0.952 |     0.134 |
             |     0.007 |     0.942 |           |
-------------|-----------|-----------|-----------|
Column Total |      1204 |       189 |      1393 |
             |     0.864 |     0.136 |           |
-------------|-----------|-----------|-----------|
```

We've lowered our false positive rate and improved our overall performance. Let's see what a different Laplace correction might do - perhaps this is a parameter we ought to fiddle with?

```
> model3 = naiveBayes(sms_train1, sms_train_labels, laplace = 0.5)
> pred3 = predict(model3,sms_test1, type="class")
> CrossTable(pred3, 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:  1393


             | Actual
   Predicted |       ham |      spam | Row Total |
-------------|-----------|-----------|-----------|
         ham |      1197 |        16 |      1213 |
             |     0.987 |     0.013 |     0.871 |
             |     0.994 |     0.085 |           |
-------------|-----------|-----------|-----------|
        spam |         7 |       173 |       180 |
             |     0.039 |     0.961 |     0.129 |
             |     0.006 |     0.915 |           |
-------------|-----------|-----------|-----------|
Column Total |      1204 |       189 |      1393 |
             |     0.864 |     0.136 |           |
-------------|-----------|-----------|-----------|
```

Not much improvement, but some. We could simply stick with the convention of Laplace = 1 since we have a relatively large sample size in this case.