Predicting a message as SPAM or non-SPAM

Rahul Singh

1/15/2018

Sources: Machine Learning with R by Brett Lantz (Second Edition); Data adapted from the SMS Spam Collection http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/

We will see how we can filter mobile SMS and classify them as Spam or not.

Classification using Bayesian methods utilize training data to calculate an observed probability of each outcome based on the evidence provided by feature values. When the classifier is applied to unlabelled data then it uses the observed probabilities to predict the most likely class.

Example of application: text classification (for spam), intrusion or anamoly detection, diagnosing medical conditions given a set of observed symptoms.

Before jumping in to the algorithm it is better to refresh your understanding about the folloeing concepts:

1. Mutually Exclusive Events
2. Exhaustive Events
3. Complement
4. Joint probability
5. Conditional probability P(A|B) = P (A B)/ P(B) P(A|B) = P(B|A) * P(A)/ P(B) Eg: In this P(A|B) is posterior probability, P(B|A) is likelihood, P(A) is prior probability and P(B) is the marginal likelihood

(likelihood vs marginal likelihood posterior probability frequency table to construct likelihood table)

6. Independent Events (Naive Bayes assumes that there is no interaction between events. But in real-life scenarios, we mostly observe that the events may not be independent)

Pros of Naive Bayes Algorithm: simple, fast, effective, works well with noisy and missing data, requires few examples for training but it also performs well with large number of examples

Cons of Naive Bayes Algorithm: relies on an often-faulty assumption of equally important and independent features, not ideal for datasets with many numerical features, estimated probabilities are less reliable than the predicted classes

For example, if you were attempting to identify spam by monitoring e-mail messages, it is almost certainly true that some features will be more important than others. For example, the e-mail sender may be a more important indicator of spam than the message text. Additionally, the words in the message body are not independent from one another, since the appearance of some words is a very good indication that other words are also likely to appear.

Laplace estimator: In practice some events never occur in the beginning, so when in conditional probability the multiplied product is zero. This may cause an error in terms of classification. A solution to this problem involves using something called the Laplace estimator, which is named after the French mathematician Pierre-Simon Laplace. The Laplace estimator essentially adds a small number to each of the counts in the frequency table, which ensures that each feature has a nonzero probability of occurring with each class. Typically, the Laplace estimator is set to 1, which ensures that each class-feature combination is found in the data at least once.

Some other points:

Categorical Inputs: Naive Bayes assumes label attributes such as binary, categorical or nominal.

Gaussian Inputs: If the input variables are real-valued, a Gaussian distribution is assumed. In which case the algorithm will perform better if the univariate distributions of your data are Gaussian or near-Gaussian. This may require removing outliers (e.g. values that are more than 3 or 4 standard deviations from the mean).

Classification Problems: Naive Bayes is a classification algorithm suitable for binary and multiclass classification.

Log Probabilities: The calculation of the likelihood of different class values involves multiplying a lot of small numbers together. This can lead to an underflow of numerical precision. As such it is good practice to use a log transform of the probabilities to avoid this underflow.

Kernel Functions: Rather than assuming a Gaussian distribution for numerical input values, more complex distributions can be used such as a variety of kernel density functions.

Update Probabilities: When new data becomes available, you can simply update the probabilities of your model. This can be helpful if the data changes frequently.

Using numeric features with Naive Bayes:

Because Naive Bayes uses frequency tables to learn the data, each feature must be categorical in order to create the combinations of class and feature values comprising of the matrix. Since numeric features do not have categories of values, the preceding algorithm does not work directly with numeric data. There are, however, ways that this can be addressed.

One easy and effective solution is to discretize numeric features, which simply means that the numbers are put into categories known as bins. For this reason, discretization is also sometimes called binning. This method is ideal when there are large amounts of training data, a common condition while working with Naive Bayes.

There are several different ways to discretize a numeric feature. Perhaps the most common is to explore the data for natural categories or cut points in the distribution of data. For example, suppose that you added a feature to the spam dataset that recorded the time of night or day the e-mail was sent, from 0 to 24 hours past midnight.

One thing to keep in mind is that discretizing a numeric feature always results in a reduction of information as the feature's original granularity is reduced to a smaller number of categories. It is important to strike a balance here. Too few bins can result in important trends being obscured. Too many bins can result in small counts in the Naive Bayes frequency table, which can increase the algorithm's sensitivity to noisy data.

```
#Importing dataset
#OpenRfine software was used to clean and convert the raw data into csv file
sms_data = read.csv("~/Desktop/Naive Bayes/SMSSpamCollection.csv")
str(sms_data) #notice that both columns are factors
```

```
## 'data.frame':    5574 obs. of  2 variables:
##  $ Column.1: Factor w/ 2 levels "ham","spam": 1 1 2 1 1 2 1 1 2 2 ...
##  $ Column.2: Factor w/ 5171 levels " &lt;#&gt;  in mca. But not conform.",..: 1121 3223 1019 4257 28
```

```
names(sms_data) <- c("Classified", "Message") #renaming the first two rows
sms_data$Message<-as.character(sms_data$Message) #converting Message column to strings (character)
```

```
table(sms_data$Classified) #ham is normal classified message and spam is unsolicited bulk classified me
```

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

```
library(tm) #to load text mining package
```

```
## Warning: package 'tm' was built under R version 3.4.3
```

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

```
library(NLP)
```

```
#We create a corpus, which is a collection of text documents
#VCorpus is a volatile corpus and is stored in memory as opposed to being stored in a disk, which is do
sms_corpus<- VCorpus(VectorSource(sms_data$Message)) #this creates a document of each message and we se
```

```
## Warning in as.POSIXlt.POSIXct(Sys.time(), tz = "GMT"): unknown timezone
## 'zone/tz/2017c.1.0/zoneinfo/Europe/Madrid'
```

```
print(sms_corpus)
```

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

```
inspect(sms_corpus[1:5]) #tm corpus is a complex list and we can get summary of specific messages using
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 5
##
## [[1]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 111
##
## [[2]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 29
##
## [[3]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 155
##
## [[4]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 49
##
## [[5]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 61
```

```
as.character(sms_corpus[[1]]) #to view specific messages in the list
```

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

```
library(base)
lapply(sms_corpus[1:2], as.character) #to view specific messages using lapply
```

```
## $`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..."
```

```
#we now transform all the characters to lowercase
sms_corpus_clean<- tm_map(sms_corpus, content_transformer(tolower))
lapply(sms_corpus_clean[1:2], as.character) #to check whether our function worked
```

```
## $`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..."
```

```
#we now strip all the numbers from the sms_corpus_clean
sms_corpus_clean<- tm_map(sms_corpus_clean, removeNumbers)
#Note that the preceding code did not use the content_transformer() function. This is because removeNum
```

```
#to remove  filler words such as to, and, but, and or from our SMS messages we use stopwords() function

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

#Since stopwords() simply returns a vector of stop words, had we chosen so, we could have replaced it w
```

```
#to eliminate any punctuation from the text messages using the built-in removePunctuation() transformat

#replacePunctuation <- function(x) {
#    gsub("[[:punct:]]+", " ", x)
#}
#we can also create a removePunctuation function to replace punctuation marks with blank space

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

```
#SnowballC package has wordStem() function to convert all words such as doing, did, etc to do... played

#to apply the wordStem() function to an entire corpus of text documents, the tm package includes a stem
sms_corpus_clean <- tm_map(sms_corpus_clean, stemDocument)
```

```
#to strip additional whitespace created by too many blank spaces
sms_corpus_clean <- tm_map(sms_corpus_clean, stripWhitespace)
as.character(sms_corpus[1])
```

```
## [1] "list(list(content = \"Go until jurong point, crazy.. Available only in bugis n great world la e
## [2] "list()"
## [3] "list()"
```

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

```
## [1] "list(list(content = \"go jurong point crazi avail bugi n great world la e buffet cine got amor w
## [2] "list()"
## [3] "list()"
```

```
#now the messages have been limited to the most interesting words, and punctuation/ capitalization have
```

Now we split text documents into words and create a Document Term Matrix (DTM) in which 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. The following illustration depicts only a small portion of

the DTM for the SMS corpus, as the complete matrix has 5,574 rows and over 7,000 columns.

The fact that each cell in the table is zero implies that none of the words listed on the top of the columns appear in any of the rst ve messages in the corpus. This highlights the reason why this data structure is called a sparse matrix; the vast majority of the cells in the matrix are lled with zeros. Stated in real-world terms, although each message must contain at least one word, the probability of any one word appearing in a given message is small.

```
#creating a DTM sparse matrix
sms_dtm <- DocumentTermMatrix(sms_corpus_clean)
```

```
#if we had not cleaned the data manually, then we can also use this code
sms_dtm2 <- DocumentTermMatrix(sms_corpus, control = list(
  tolower = TRUE,
  removeNumbers = TRUE,
  stopwords = TRUE,
  removePunctuation = TRUE,
  stemming = TRUE
))
```

```
#comparing the two cleaning methods manual vs automatic
print(sms_dtm)
```

```
## <<DocumentTermMatrix (documents: 5574, terms: 6605)>>
## Non-/sparse entries: 42624/36773646
## Sparsity           : 100%
## Maximal term length: 40
## Weighting          : term frequency (tf)
```

```
print(sms_dtm2)
```

```
## <<DocumentTermMatrix (documents: 5574, terms: 7015)>>
## Non-/sparse entries: 43735/39057875
## Sparsity           : 100%
## Maximal term length: 40
## Weighting          : term frequency (tf)
```

```
#The reason for this discrepancy has to do with a minor difference in the ordering of the preprocessing
```

```
#creating test and train dataset
sms_dtm_train <- sms_dtm[1:4460, ]
sms_dtm_test  <- sms_dtm[4461:5574, ]
```

```
#For convenience later on, it is also helpful to save a pair of vectors with labels for each of the row
sms_train_labels <- sms_data[1:4460, ]$Classified
sms_test_labels  <- sms_data[4461:5574, ]$Classified
```

```
#to confirm that the subsets are representative of the complete set of SMS data, we compare the proport
prop.table(table(sms_train_labels))
```

```
## sms_train_labels
##       ham      spam
## 0.8650224 0.1349776
```

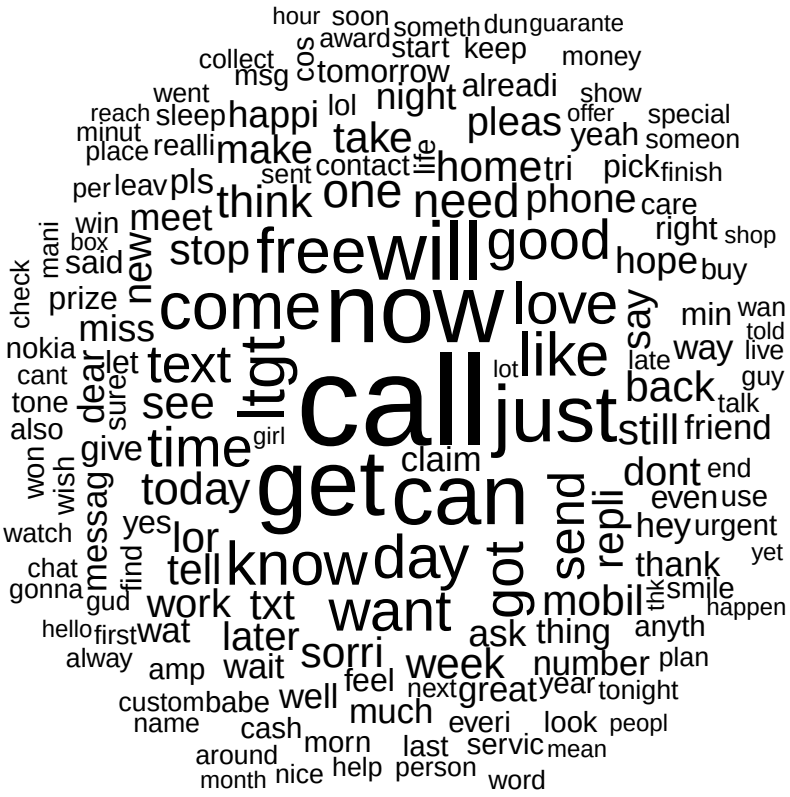
```
prop.table(table(sms_test_labels))
```

```
## sms_test_labels
##       ham      spam
## 0.8698384 0.1301616
```

```
#We load wordcloud library to visualize types of words in SMS messages
library(wordcloud)
```

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

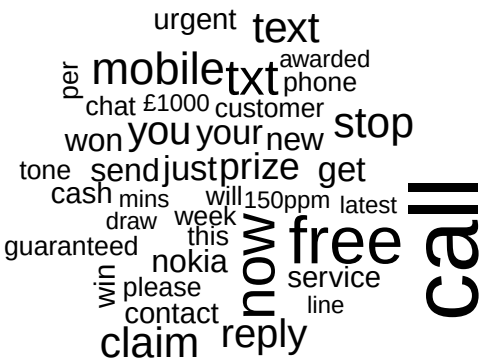
```
library(RColorBrewer)
wordcloud(sms_corpus_clean, min.freq = 50, random.order = FALSE)
```




```
#Since we specified random.order = FALSE, the cloud will be arranged in a non random order with higher
```

We now try to analyze word clouds for spam and ham by subsetting the two into new data by SMS type classified.

```
spam<-subset(sms_data, Classified == "spam")
ham<-subset(sms_data, Classified == "ham")
wordcloud(spam$Message, max.words = 40, scale = c(3, 0.5))
```




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

Creating indicator features for frequent words

We transform the sparse matrix into a data structure that can be used to train a Naive Bayes classifier. Currently, the sparse matrix includes over 6,500 features; this is a feature for every word that appears in at least one SMS message. It's unlikely that all of these are useful for classification. To reduce the number of features, we will eliminate any word that appears in less than five SMS messages, or in less than about 0.1 percent of the records in the training data.

Finding frequent words requires use of the findFreqTerms() function in the tm package. This function takes a DTM and returns a character vector containing the words that appear for at least the specified number of times. For instance, the following command will display the words appearing at least ve times in the sms_dtm_train matrix findFreqTerms(sms_dtm_train, 5)

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

```
##  chr [1:1211] "£wk" "abiola" "abl" "abt" "accept" "access" "account" ...
```

We now need to filter our DTM to include only the terms appearing in a specified vector. As done earlier, we'll use the data frame style [row, col] operations to request specific portions of the DTM, noting that the columns are named after the words the DTM contains. We can take advantage of this to limit the DTM to specific words. We want all the rows, but only the columns representing the words in the sms_freq_words vector.

```
sms_dtm_freq_train<- sms_dtm_train[ , sms_freq_words]
sms_dtm_freq_test <- sms_dtm_test[ , sms_freq_words]
```

The training and test datasets now include 1,211 features, which correspond to words appearing in at least five messages.

The Naive Bayes classifier is typically trained on data with categorical features. This poses a problem, since the cells in the sparse matrix are numeric and measure the number of times a word appears in a message. We need to change this to a categorical variable that simply indicates yes or no depending on whether the word appears at all.

```
#we define a function to convert counts to Yes or No strings
convert_counts <- function(x) {
    x <- ifelse(x > 0, "Yes", "No")
}

#We now need to apply convert_counts() to each of the columns in our sparse matrix.
```

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

The result will be two character type matrixes, 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.

Training the model using e1071 package of Naive Bayes

```
library(e1071)
sms_classifier <- naiveBayes(sms_train, sms_train_labels)
#naiveBayes(training dataframe, factor vector for classification,laplace is 0 by default); this functio

sms_test_pred <- predict(sms_classifier, sms_test)
#this prediction function takes in predict(model data used for training, test data frame, type ="class"
```

Confusion Matrix: Checking Model Performance

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

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |-------------------------|
##
##
## Total Observations in Table:  1114
##
##
##              | actual
##    predicted |       ham |      spam | Row Total |
## -------------|-----------|-----------|-----------|
##          ham |       963 |        15 |       978 |
##              |     0.985 |     0.015 |     0.878 |
##              |     0.994 |     0.103 |           |
## -------------|-----------|-----------|-----------|
##         spam |         6 |       130 |       136 |
##              |     0.044 |     0.956 |     0.122 |
##              |     0.006 |     0.897 |           |
## -------------|-----------|-----------|-----------|
## Column Total |       969 |       145 |      1114 |
##              |     0.870 |     0.130 |           |
## -------------|-----------|-----------|-----------|
##
##
```

Looking at the table we observe that there are only 15 + 6 =21 of the 1114 SMS messages that were incorrectly classified

Improving the model by putting laplace value as 1

```r
sms_classifier2 <- naiveBayes(sms_train, sms_train_labels, laplace = 1)
sms_test_pred2 <- predict(sms_classifier2, sms_test)
CrossTable(sms_test_pred2, sms_test_labels,
    prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,
    dnn = c('predicted', 'actual'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Col Total |
## |-------------------------|
##
##
## Total Observations in Table:  1114
##
##
##              | actual
##    predicted |       ham |      spam | Row Total |
## -------------|-----------|-----------|-----------|
##          ham |       964 |        18 |       982 |
##              |     0.995 |     0.124 |           |
## -------------|-----------|-----------|-----------|
##         spam |         5 |       127 |       132 |
##              |     0.005 |     0.876 |           |
## -------------|-----------|-----------|-----------|
## Column Total |       969 |       145 |      1114 |
##              |     0.870 |     0.130 |           |
## -------------|-----------|-----------|-----------|
##
##
```

In this case we observe that laplace is not helping us to achieve higher accuracy of the model.

Conclusion: We observe that our initial model is better able to classify SMS messages as SPAM or HAM; the Naive Bayes classifier is often used for text classification.