Naive Bayes

By: Karan Daiya

Northeastern University: College of Professional Studies

NUID: 001083274

Abstract

These days advertisers use short message services to attract their audience with unwanted advertisements known as SMS spam. Naïve Bayes method has been pretty successful for email spams so here in Part A, I have selected a dataset of Spam SMS and in Part B the dataset contains public reviews and other column is for positive and negative type. I have modified the original dataset to make it simple and better for this assignment.

Part A

Step 1: Collecting data.

The dataset contains text of SMS messages and a label row indicating the unwanted messages. Legitimate messages are labelled with "ham" and the unwanted are labelled "spam".

Step 2: Exploring and preparing the data.

sms_raw <- read.csv("/Users/karan/Downloads/sms_spam.csv", stringsAsFactors = FALSE)

str(sms_raw)

Above code will read the csv file from desired path and using str() function we can see that there are 5574 observations of 2 variables.

Output:

```
'data.frame':	5574 obs. of  2 variables:
 $ type: chr  "ham" "ham" "spam" "ham" ...
 $ text: chr  "Go until jurong point, crazy.. Available only in bugis n great world la
e buffet... Cine there got amore wat..." "Ok lar... Joking wif u oni..." "Free entry
in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive
entry question("| __truncated__ "U dun say so early hor... U c already then say..."
...
```

As it is the categorical value, I converted it into factors,
sms_raw$type <- factor(sms_raw$type)
str(sms_raw$type)
table(sms_raw$type)
Output:

```
Factor w/ 2 levels "ham","spam": 1 1 2 1 1 2 1 1 2 2 ...

 ham spam
4827  747
```

From the output we can see that there are 747 messages which are spam and the remaining are labeled ham.

Data Preparation: processing text data for analysis.
For text mining, install library tm by writing install.packages("tm") and load it by running library(tm) command. Corpus() function creates an R object to store text documents. Corpus means collection of text documents.

```{r Creating a corpus}
sms_corpus <- Corpus(VectorSource(sms_raw$text))
print(sms_corpus)
```

The above code will create a corpus function which will be stored in sms_corpus.

```{r}
corpus_clean <- tm_map(sms_corpus, tolower)
corpus_clean <- tm_map(corpus_clean, removeNumbers)
```

Above code will remove the numbers and convert uppercase characters to lowercase.

```{r}
corpus_clean <- tm_map(corpus_clean, removeWords, stopwords())
corpus_clean <- tm_map(corpus_clean, removePunctuation)
corpus_clean <- tm_map(corpus_clean, stripWhitespace)
```

Above code will remove all the unnecessary filler words, punctuation marks and white space respectively. The tm package provides functionality to tokenize the SMS message corpus. The DocumentTermMatrix() function will take a corpus and make a sparse matrix in which rows will indicate documents i.e. messages and the column will indicate each words. Each cell in a sparse matrix will contain the number of frequency the word will appear in that desired message.

```{r Creating a sparse matrix}
sms_dtm <- DocumentTermMatrix(corpus_clean)
View(sms_dtm)
```

Creating training and test dataset:

```{r}

sms_raw_train <- sms_raw[1:4169, ]

sms_raw_test <- sms_raw[4170:5559, ]

sms_dtm_train <- sms_dtm[1:4169, ]

sms_dtm_test <- sms_dtm[4170:5559, ]

sms_corpus_train <- corpus_clean[1:4169]

sms_corpus_test <- corpus_clean[4170:5559]

```

```{r Comparing the spam proportion}

prop.table(table(sms_raw_train$type))

prop.table(table(sms_raw_test$type))

```

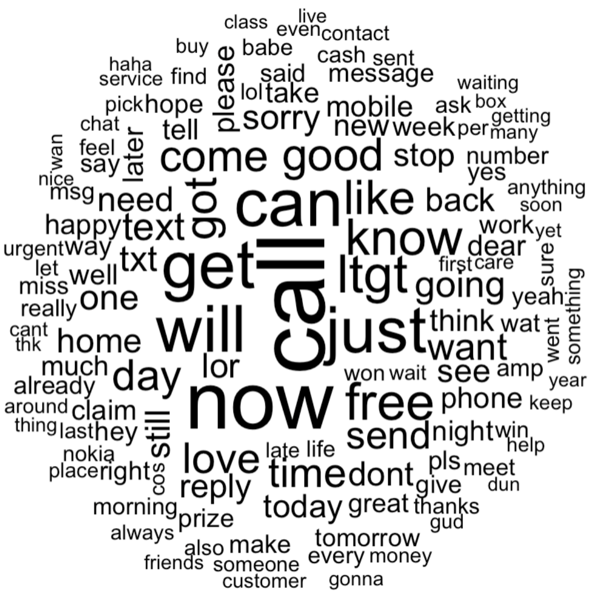
```
ham       spam
0.8647158 0.1352842

      ham      spam
0.8697842 0.1302158
```

Here, I have used 75% of my data for training and remaining 25% for testing. As the dataset is already set to random, I took first 4169 rows for training and remaining 1390 for test. After comparing, we can say that in both test and training data, 86% are ham messages and remaining 14% are spam messages.

Visualizing text data: Word Clouds

It is a way of representing the words with most frequency in data. A word cloud can be created directly from a tm corpus object.

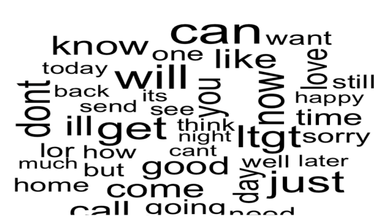
```{r Creating a word cloud}

wordcloud(sms_corpus_train, min.freq = 40, random.order = FALSE)

```



Now let's use subset() function,

```{r}

spam <- subset(sms_raw_train, type == "spam")

ham <- subset(sms_raw_train, type == "ham")

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

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

View(spam)

```

The above R code will create subsets and then will make wordcloud. Output is shows below,



```{r}
findFreqTerms(sms_dtm_train, 5)
sms_dict <- c(findFreqTerms(sms_dtm_train, 5))
```

This code will create indicator features for frequent world. findFreqTerms() function is used for finding frequent words.

```{r}
sms_train <- DocumentTermMatrix(sms_corpus_train, list(dictionary = sms_dict))
sms_test <- DocumentTermMatrix(sms_corpus_test, list(dictionary = sms_dict))
```

Above code is used for limiting our training and test matrixes to only the words in the preceding dictionary. The naive Bayes classifier is typically trained on data with categorical features. The sparse matrix indicates the counts of words appeared in a desired message so to convert them into factors, following code is used,

```{r}
convert_counts<-function(x){
x<-ifelse(x>0,1,0)
x<-factor(x,levels=c(0,1),labels=c("No","Yes"))
return(x)
}
```

ifelse(x>0,1,0) states that if the value of x is greater than 0, it will be replaced with 1, otherwise it will remain at 0. Now, to apply convert_counts to each of the columns in sparse matrix, following code is used,

```{r}
```

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

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

```

**Step 3**: Training a model on the data.

To implement naïve bayes method, install "e1071" package and load the library.

To build our model on the sms_train matrix, following code is used,

```{r}

sms_classifier <- naiveBayes(sms_train, sms_raw_train$type)

```

Step 4: Evaluating model performance.

sms_test_pred <- predict(sms_classifier, sms_test)

The predict() function is used to predict the further values and I stored it to sms_test_pred. CrossTable() is used to compare the actual values to the predicted values. CrossTable is accessible from library "gmodel".

```{r}

library(gmodels)

CrossTable(sms_test_pred, sms_raw_test$type,

prop.chisq = FALSE, prop.t = FALSE, dnn = c('predicted', 'actual'))

```

Output:

```
Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|-------------------------|
```
```

```
Total Observations in Table:  1390

             | actual
   predicted |       ham |      spam | Row Total |
-------------|-----------|-----------|-----------|
         ham |      1203 |        28 |      1231 |
             |     0.977 |     0.023 |     0.886 |
             |     0.995 |     0.155 |           |
-------------|-----------|-----------|-----------|
        spam |         6 |       153 |       159 |
             |     0.038 |     0.962 |     0.114 |
             |     0.005 |     0.845 |           |
-------------|-----------|-----------|-----------|
Column Total |      1209 |       181 |      1390 |
             |     0.870 |     0.130 |           |
-------------|-----------|-----------|-----------|
```

Here, we can observe that there are 6 out of 1207 ham messages that were incorrectly specified as spam and 28 out of 181 spam messages were incorrectly specified as ham messages.

Step 5: Improving model performance,

Laplace allows words that appeared in zero spam or zero ham messages to have an indisputable say in the classification process.

```{r}
sms_classifier2 <- naiveBayes(sms_train, sms_raw_train$type,
laplace = 0.1)
sms_test_pred2 <- predict(sms_classifier2, sms_test)
CrossTable(sms_test_pred2, sms_raw_test$type, prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE, dnn = c('predicted', 'actual'))
```

I improved the model’s performance by building a naive bayes model as before, but this time setting laplace to 0.1 which is giving me the best accuracy which is nearly more than 90% and even better than the accuracy mentioned in the book’s example.

```
Cell Contents
|-------------------------|
|                       N |
|           N / Col Total |
|-------------------------|
Total Observations in Table:  1390

             | actual
   predicted |       ham |      spam | Row Total |
-------------|-----------|-----------|-----------|
         ham |      1207 |        26 |      1233 |
             |     0.998 |     0.144 |           |
-------------|-----------|-----------|-----------|
        spam |         2 |       155 |       157 |
             |     0.002 |     0.856 |           |
-------------|-----------|-----------|-----------|
Column Total |      1209 |       181 |      1390 |
             |     0.870 |     0.130 |           |
-------------|-----------|-----------|-----------|
```

Model Accuracy: (0.998 + 0.856)/2 = 0.927 = 92.7%

Part B

Step 1: Collecting data.

The dataset contains 1000 reviews of Amazon and a label row indicating whether the comment is positive about the product or negative about the product. Positive comments are labelled with “1” and the negative are labelled “0”.

Step 2: Exploring and preparing the data.

review <- read.csv("/Users/karan/Downloads/amazon_reviews.csv", stringsAsFactors = FALSE)

Above code will read the csv file,

```{r}

review$Type <- factor(review$Type, levels = c(0, 1), labels = c("Negative","Positive"))

```

Setting ‘1’ as ‘positive’ and ‘0’ as ‘negative’ for our easy reference. As it is the categorical value, I converted it into factors,

```{r}
review$Type <- factor(review$Type)
str(review$Type)
table(review$Type)
```

Output:

```
Factor w/ 2 levels "Negative","Positive": 1 2 2 1 2 1 1 2 1 1 ...

Negative Positive
     500      500
```

From this output we can say that there are equal amount of negative and positive reviews.

Data Preparation: processing text data for analysis.

For text mining, install library tm by writing install.packages("tm") and load it by running library(tm) command. Corpus() function creates an R object to store text documents. Corpus means collection of text documents.

```{r Creating a corpus}

review_corpus <- Corpus(VectorSource(review$Reviews))
```

```
print(review_corpus)
```

Output:

```
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 1000
```

The above code will create a corpus function which will be stored in review_corpus.

```{r}
review_clean <- tm_map(review_corpus, tolower)
review_clean <- tm_map(review_clean, removeNumbers)
```

tm_map() function provides a method for transforming a tm corpus and here I am removing numbers and converting the letters to lowercase.

```{r}
review_clean <- tm_map(review_clean, removeWords, stopwords())
review_clean <- tm_map(review_clean, removePunctuation)
review_clean <- tm_map(review_clean, stripWhitespace)
```

Above code will remove all the unwanted filler words, punctuations and white space respectively.

The DocumentTermMatrix() function will take a corpus and make a sparse matrix in which rows will indicate documents i.e. reviews, the column will indicate each words. Each cell in a sparse matrix will contain the number of frequency the word will appear in that desired comment.

```{r Creating a sparse matrix}
review_dtm <- DocumentTermMatrix(review_clean)
View(review_dtm)
str(review)
```

Output:

```
'data.frame':	1000 obs. of  7 variables:
 $ Reviews: chr  "

So there is no way for me to plug it in here in the US unless I go by a converter."
"Good case" "Great for the jawbone." "Tied to charger for conversations lasting more
than 45 minutes.MAJOR PROBLEMS!!" ...
 $ X      : int  0 NA 1 0 1 0 NA 1 NA 0 ...
 $ X.1    : int  NA 1 NA NA NA NA 0 NA 0 NA ...
 $ X.2    : int  NA NA NA NA NA NA NA NA NA NA ...
 $ X.3    : int  NA NA NA NA NA NA NA NA NA NA ...
 $ X.4    : int  NA NA NA NA NA NA NA NA NA NA ...
```

```
 $ Type   : Factor w/ 2 levels "Negative","Positive": 1 2 2 1 2 1 1 2 1 1 ...
```

As we can see that there are 1000 observations of 7 variables in this data frame.

```{r}
review_raw_train <- review[1:749, ]
review_raw_test <- review[750:1000, ]
review_dtm_train <- review_dtm[1:749, ]
review_dtm_test <- review_dtm[750:1000, ]
review_corpus_train <- review_clean[1:749]
review_corpus_test <- review_clean[750:1000]
```

Above code is for creating training and test dataset. Here I have used 75 percent for training and 25 percent for testing purpose.

```{r Comparing the spam proportion}
prop.table(table(review_raw_train$Type))
prop.table(table(review_raw_test$Type))
```

Output:

```
Negative  Positive
0.4873164 0.5126836

 Negative  Positive
0.5378486 0.4621514
```

Here we can see that there are 48% negative comments in training dataset and 53% in test dataset.

Creating a word cloud,

```{r}

wordcloud(review_corpus_train, min.freq = 30, random.order = FALSE)

```

To access this function, installing wordcloud library is necessary.

Output:

```{r}
positive <- subset(review_raw_train, Type == "Positive")
negative <- subset(review_raw_train, Type == "Negative")
wordcloud(positive$Reviews, max.words = 40, scale = c(3, 0.5))
wordcloud(negative$Reviews, max.words = 40, scale = c(3, 0.5))
```

Above code will first make subsets of positive and negative reviews and then plots the word cloud.

Output:


```{r}
findFreqTerms(review_dtm_train, 5)
review_dict <- c(findFreqTerms(review_dtm_train, 5))
```

findFreqTerms() function is used for finding the frequent words appearing the desired times and it is stored in review_dist.

```{r}
review_train <- DocumentTermMatrix(review_corpus_train, list(dictionary = review_dict))
review_test <- DocumentTermMatrix(review_corpus_test, list(dictionary = review_dict))
```

Above code is for limiting our training and test matrixes to only the words in the preceding dictionary. The naive Bayes classifier is typically trained on data with categorical features. The sparse matrix indicates the counts of words appeared in a desired message so to convert them into factors, following code is used,

```{r}
convert_counts<-function(y){
y<-ifelse(y>0,1,0)
y<-factor(y,levels=c(0,1),labels=c("No","Yes"))
return(y)
}
```

ifelse(x>0,1,0) states that if the value of x is greater than 0, it will be replaced with 1, otherwise it will remain at 0. Now, to apply convert_counts to each of the columns in sparse matrix, following code is used,

```{r}

review_train <- apply(review_train, MARGIN = 2, convert_counts)

review_test <- apply(review_test, MARGIN = 2, convert_counts)
```

Step 3: Training a model on the data.

To implement naïve bayes method, install "e1071" package and load the library.

To build our model on the sms_train matrix, following code is used,

```{r}

review_classifier <- naiveBayes(review_train, review_raw_train$Type)
```

Step 4: Evaluating model performance.

review_test_pred <- predict(review_classifier, review_test)

The predict() function is used to predict the further values and I stored it to sms_test_pred.

CrossTable() is used to compare the actual values to the predicted values. CrossTable is accessible from library "gmodel".

```{r}

library(gmodels)

CrossTable(review_test_pred, review_raw_test$Type,

prop.chisq = FALSE, prop.t = FALSE, dnn = c('predicted', 'actual'))

```

Output:

```
Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|-------------------------|


Total Observations in Table:  251


             | actual
   predicted |  Negative |  Positive | Row Total |
-------------|-----------|-----------|-----------|
    Negative |       110 |        39 |       149 |
             |     0.738 |     0.262 |     0.594 |
             |     0.815 |     0.336 |           |
-------------|-----------|-----------|-----------|
    Positive |        25 |        77 |       102 |
             |     0.245 |     0.755 |     0.406 |
             |     0.185 |     0.664 |           |
-------------|-----------|-----------|-----------|
Column Total |       135 |       116 |       251 |
             |     0.538 |     0.462 |           |
-------------|-----------|-----------|-----------|
```

Here, we can observe that there are 25 out of 135 negative messages that were incorrectly specified as positive and 39 out of 116 positive messages were incorrectly specified as negative messages.

Step 5: Improving model performance,

Laplace allows words that appeared in zero positive or zero negative messages to have an indisputable say in the classification process.

```{r}
review_classifier2 <- naiveBayes(review_train, review_raw_train$Type,
laplace = 0.1)
review_test_pred2 <- predict(review_classifier2, review_test)
CrossTable(review_test_pred2, review_raw_test$Type, prop.chisq = FALSE, prop.t = FALSE,
prop.r = FALSE, dnn = c('predicted', 'actual'))
```

Above code is for improving model performance by building a naive bayes model as before, but this time setting laplace to 0.1 which is giving me accuracy around 74% which got improved from 73%(previous).

Output:

```
Cell Contents
|-------------------------|
|                       N |
|           N / Col Total |
|-------------------------|


Total Observations in Table:  251


             | actual
   predicted |  Negative |  Positive | Row Total |
-------------|-----------|-----------|-----------|
    Negative |       111 |        39 |       150 |
             |     0.822 |     0.336 |           |
-------------|-----------|-----------|-----------|
    Positive |        24 |        77 |       101 |
             |     0.178 |     0.664 |           |
-------------|-----------|-----------|-----------|
Column Total |       135 |       116 |       251 |
             |     0.538 |     0.462 |           |
```

Here, we can observe that there are 24 out of 135 negative messages that were incorrectly specified as positive and 39 out of 116 positive messages were incorrectly specified as negative messages.

Model Accuracy: (0.822 + 0.664)/2 = 0.743 = 74.3%

Summary

Ultimately, the Part A model was able to classify nearly 98 percent of all SMS messages correctly as spam or ham and Part B model was able to classify neary 82.22 percent of all reviews correctly as positive or negative.

References:

Machine Learning with R - Second Edition. Retrieved from

https://edu.kpfu.ru/pluginfile.php/278552/mod_resource/content/1/MachineLearningR__Brett_Lantz.pdf

Sentiment Labelled Sentences Dataset. Retrieved from,

https://www.kaggle.com/marklvl/sentiment-labelled-sentences-data-set