Sean Toon

IT-460-18500-M01 Machine Learning

October 13, 2024

Module Six Lab Assignment: Naïve Bayes

For this project, I learned how to use the Naïve Bayes algorithm. I used a dataset of SMS messages to try and filter out spam messages from ham messages. The first step in the project was to evaluate the data and prepare it accordingly so that I could use it for my algorithm. One of the challenges with the data was that all the text was in SMS form, which is hard for a computer to understand. So, I used the following code to transform the data into a representation known as a bag-of-words. The representation ignores word order and provides a variable to indicate whether the word appears at all. I started by importing the data and then saving it to a data frame with:

```
> sms_raw <- read.csv("C:\\Users\\toons\\Downloads\\sms_spam.csv", stringsAsFactors = FALSE)
> str(sms_raw)
'data.frame':   5559 obs. of  2 variables:
 $ type: chr  "ham" "ham" "ham" "spam" ...
 $ text: chr  "Hope you are having a good week. Just checking in" "K..give back my thanks." "Am also doing in cbe
```

Then, I used the factor function to convert the "type" element into a factor since it is a categorical variable with the following code and also used the table function to ensure they were properly converted:

```
> sms_raw$type <- factor(sms_raw$type)
> str(sms_raw$type)
 Factor w/ 2 levels "ham","spam": 1 1 1 2 2 1 1 1 2 1 ...
> table(sms_raw$type)

 ham spam
4812  747
>
```

I then used a text mining package titled tm to clean the data of things such as punctuation, numbers and uninteresting words. I also created a corpus, or a collection of text documents and in this case, SMS messages. I used the VCorpus() function below, the v stands for volatile memory. I also used the VectorSource reader function to create a source object from the sms_raw$text vector:

```
> library(tm)
Loading required package: NLP
> sms_corpus <- VCorpus(VectorSource(sms_raw$text))
> print(sms_corpus)
<<VCorpus>>
Metadata:  corpus specific: 0, document level (indexed): 0
Content:  documents: 5559
```

I used the inspect() function, the as.character() function, and the lapply() function to help read and select the documents in the corpus:

```
> inspect(sms_corpus[1:2])
<<VCorpus>>
Metadata:  corpus specific: 0, document level (indexed): 0
Content:  documents: 2

[[1]]
<<PlainTextDocument>>
Metadata:  7
Content:  chars: 49

[[2]]
<<PlainTextDocument>>
Metadata:  7
Content:  chars: 23

> as.character(sms_corpus[[1]])
[1] "Hope you are having a good week. Just checking in"
>  lapply(sms_corpus[1:2], as.character)
$`1`
[1] "Hope you are having a good week. Just checking in"

$`2`
[1] "K..give back my thanks."
```

I used the tm_map() function, the tm wrapper function content_transformer(), and the tolower() function to transform the data into individual words and in a way so that the text is standardized and saved the result to an object “corpus_clean”:

```
>  sms_corpus_clean <- tm_map(sms_corpus,
+     content_transformer(tolower))
> as.character(sms_corpus[[1]])
[1] "Hope you are having a good week. Just checking in"
> as.character(sms_corpus_clean[[1]])
[1] "hope you are having a good week. just checking in"
>
```

In the example I can see that the code worked.

Now to remove the numbers, I can simply use the tm_map() function again but with the removeNumbers parameter:

```
> sms corpus clean <- tm map(sms corpus clean, removeNumbers)
```

To remove the filler words or stop words, I used the stopwords() function and the removeWords() function, and the tm_map() function in the following way:

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

Next, to remove the punctuation, I used the tm_map() function again with the removePunctuation() transformation:

```
>  sms_corpus_clean <- tm_map(sms_corpus_clean, removePunctuation)
> removePunctuation("hello...world")
[1] "helloworld"
```

I was also able to transform words in to the root form through the process of stemming. I used the stemDocument() function from the SnowballC package and then tested out the function. For my last step I removed any left over white spaces using the stripWhitespace() function:

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

After cleaning the data I used the process of tokenization to split the messages into individual terms. I started by creating a document-term matrix(DTM) which stores a number that indicates the number of times the word represented by the column shows up in the document represented by the row. I used the following code to create the matrix and store it in the sms_dtm object:

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

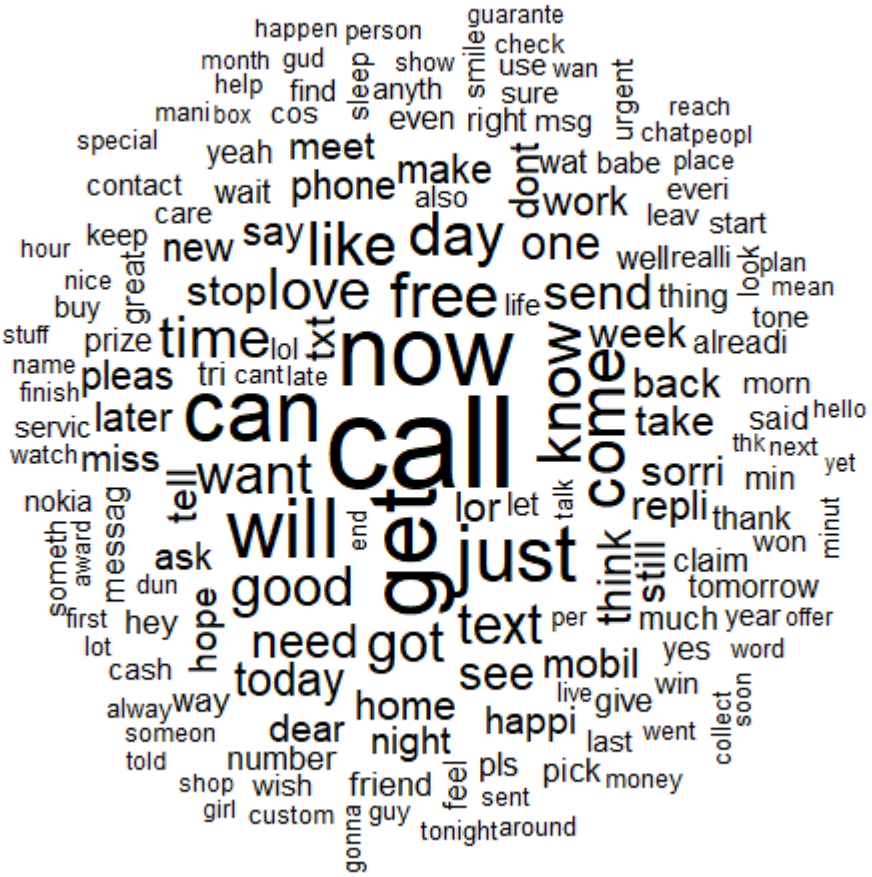
Next, I created training and test data sets. The training set represents 75 percent of the data and the test data represents the remaining 25 percent. I simply requested a specific range of rows and columns to split the data with the following code:

```
> sms_dtm_train <- sms_dtm[1:4169, ]
> sms_dtm_test  <- sms_dtm[4170:5559, ]
```

To help visualize the data, I used the wordcloud package to create a word cloud diagram. I used the following code and parameters, min.freq (number of times the word needs to show up in the data to be displayed in the diagram), and random.order (to randomize the arrangement of words:

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

This was the resulting diagram:



The last step for preparing the data that I took was to find the most frequent words for better classification. I eliminated any word that appears in less than five messages, or in less than about 0.1 percent of records in the training data.

I used the findFreqTerms() function with the sms_dtm_train matrix:

```
> findFreqTerms(sms_dtm_train, 5)
```

And then saved the words frequent words in character vector:

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

```
> str(sms_freq_words)
 chr [1:1137] "£wk" "abiola" "abl" "abt" "accept" "access" "account" "across" "act" "activ" "actual" "add" ...
```

I then used the following commands to filter the DTM to include only terms appearing in the frequent word vector in the [row,col] data frame style:

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

Then, because the naïve bayes classifier is trained on data with categorical features, I used the following code to change the numeric values of frequent word to a simple yes or no of whether it appeared or not and applied it to the training and test sets:

```
> sms_dtm_freq_test <- sms_dtm_test[ , sms_freq_words]
> 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)
```

Finally, for the last part of the project, I was able to use the transformed data to apply the Naïve Bayes algorithm. I used the e1071 package. To build the model on the sms_train matrix, I used the simple command:

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

Then, it was time to evaluate the performance of the model. I used the object sms_classifier to generate predictions with the predict() function:

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

And then compared the results to the true values using the CrossTable() function in the gmodels package:

```
> library(gmodels)
> CrossTable(sms_test_pred, sms_test_labels,
+     prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
+     dnn = c('predicted', 'actual'))


   Cell Contents
|-------------------------|
|                       N |
|         N / Table Total |
|-------------------------|

Total Observations in Table:  1390


             | actual
   predicted |       ham |      spam | Row Total |
-------------|-----------|-----------|-----------|
         ham |      1201 |        30 |      1231 |
             |     0.864 |     0.022 |           |
-------------|-----------|-----------|-----------|
        spam |         6 |       153 |       159 |
             |     0.004 |     0.110 |           |
-------------|-----------|-----------|-----------|
Column Total |      1207 |       183 |      1390 |
-------------|-----------|-----------|-----------|
```

As an experiment for the end of the lab, I used the Naïve Bayes function from the klaR package instead. I also used the caret package to see if automatically tuning the laplace and usekernel parameters for the algorithm would increase the accuracy from previous attempts. I used values of TRUE and FALSE for the usekernal parameters and values 1-5 for the laplace parameters and the changes did not increase or decrease the accuracy at all.

```
> library(caret)
> library(klaR)
>
> tune_grid <- expand.grid(
+   laplace = 0:5,
+   usekernel = c(TRUE, FALSE),
+   adjust = 1
+ )
>
> set.seed(123)
> naive_bayes_model_tuned <- train(
+   type ~ .,
+   data = sms_train_df,
+   method = "naive_bayes",
+   tuneGrid = tune_grid,
+   trControl = trainControl(method = "cv", number = 10)
+ )
>
> print(naive_bayes_model_tuned)
Naive Bayes

4169 samples
1163 predictors
   2 classes: 'ham', 'spam'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 3751, 3751, 3753, 3753, 3751, 3753, ...
Resampling results across tuning parameters:

  laplace  usekernel  Accuracy   Kappa
  0        FALSE      0.1352828  0
  0         TRUE      0.8647172  0
  1        FALSE      0.1352828  0
  1         TRUE      0.8647172  0
  2        FALSE      0.1352828  0
  2         TRUE      0.8647172  0
  3        FALSE      0.1352828  0
  3         TRUE      0.8647172  0
  4        FALSE      0.1352828  0
  4         TRUE      0.8647172  0
  5        FALSE      0.1352828  0
  5         TRUE      0.8647172  0

Tuning parameter 'adjust' was held constant at a value of 1
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were laplace = 0, usekernel = TRUE and adjust = 1.
```

References

Lantz, B. (2019). *Machine Learning with R* (3rd ed.). Packt Publishing.

https://mbsdirect.vitalsource.com/books/9781788291552