Data 607 - Project 4 - Document Classification

William Jasmine

2022-11-17

Introduction

This document will outline the process by which a Naive Bayes classifier can be used to categorize documents as spam or ham (not spam). The data being used in this case comes from Kaggle and consists of a set of 5,574 SMS text messages. The Kaggle dataset page lists the source of the text messages, all of which are listed below:

- A collection of 425 SMS spam messages extracted from the http://co.uk-www.com/grumbletext.co.uk. This is a UK forum in which cell phone users make public claims about SMS spam messages.
- A subset of 3,375 SMS randomly chosen ham messages of the NUS SMS Corpus (NSC), which is a dataset of about 10,000 legitimate messages collected for research at the Department of Computer Science at the National University of Singapore.
- A list of 450 SMS ham messages collected from Caroline Tag's PhD Thesis.
- The SMS Spam Corpus v.0.1 Big. It contains 1,002 SMS ham messages and 322 spam messages.

The data set has also been loaded to Github page for this project.

The goal in this case is to build a classifier that can accurately determine which of the text messages in our data set are spam.

Data Import

The cell below imports the data from Github and loads it into the R dataframe, `texts`:

```
link <- getURL('https://raw.githubusercontent.com/williamzjasmine/CUNY_SPS_DS/master/DATA_607/Projects/
texts <- read_csv(link)
```

```
## New names:
## Rows: 5572 Columns: 5
## -- Column specification
## -------------------------------------------------------- Delimiter: "," chr
## (5): v1, v2, ...3, ...4, ...5
## i Use `spec()` to retrieve the full column specification for this data. i
## Specify the column types or set `show_col_types = FALSE` to quiet this message.
## * `` -> `...3`
## * `` -> `...4`
## * `` -> `...5`
```

```
glimpse(texts)
```

```
## Rows: 5,572
## Columns: 5
## $ v1   <chr> "ham", "ham", "spam", "ham", "ham", "spam", "ham", "ham", "spam",~
## $ v2   <chr> "Go until jurong point, crazy.. Available only in bugis n great w~
## $ ...3 <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ ...4 <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ ...5 <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
```

We see from the output above that the data consists of 5 columns and the expected 5,572 texts. The first and second columns (`v1` and `v2`) appear to contain the relevant information, namely the category (spam/ham) and content of the text, respectively.

Data Preparation

Cleaning The `texts` Dataframe

The last three columns in the data (`...3`, `...4`, and `..5`) appear to not contain any useful information, but the cell below checks to make sure by printing out any instances in which those columns contain non `NA` values:

```
texts %>% filter(!is.na(...3) | !is.na(...4) | !is.na(...5)) %>%
  select(...3, ...4, ...5)
```

```
## # A tibble: 50 x 3
##    ...3                                                                ...4  ...5
##    <chr>                                                               <chr> <chr>
##  1 "PO Box 5249"                                                       "MK1~  <NA>
##  2 "the person is definitely special for u..... But if the person i~ "why~ "jus~
##  3 "HOWU DOIN? FOUNDURSELF A JOBYET SAUSAGE?LOVE JEN XXX\\\"\""         <NA>  <NA>
##  4 "wanted to say hi. HI!!!\\\" Stop? Send STOP to 62468\""             <NA>  <NA>
##  5 "this wont even start........ Datz confidence..\""                   <NA>  <NA>
##  6 "PO Box 5249"                                                       "MK1~  <NA>
##  7 "GN"                                                                "GE"  "GNT~
##  8 ".;-):-D\""                                                          <NA>  <NA>
##  9 "just been in bedbut mite go 2 thepub l8tr if uwana mt up?loads ~   <NA>  <NA>
## 10 "bt not his girlfrnd... G o o d n i g h t . . .@\""                  <NA>  <NA>
## # ... with 40 more rows
```

There only exist 50 rows in which these columns are not empty, and from a quick glance it appears that they add no additional information. As such, they can be removed. This is performed in the cell below along with a number of other data cleaning steps (described in the comments):

```
# remove ...3, ...4, and ...5 columns
texts <- texts %>%
  select(v1, v2)

# give appropriate names to the remaining two columns
colnames(texts) <- c('category', 'text')

# transform spam --> 1 and ham --> 0
texts <- texts %>%
  mutate(category = ifelse(category == 'spam', 1, 0))

# add a text_id field so every text has a unique identifier
texts$text_id <- 1:nrow(texts)
```

```
head(texts)
```

```
## # A tibble: 6 x 3
##   category text                                                           text_id
##      <dbl> <chr>                                                            <int>
## 1        0 Go until jurong point, crazy.. Available only in bugis n gre~       1
## 2        0 Ok lar... Joking wif u oni...                                       2
## 3        1 Free entry in 2 a wkly comp to win FA Cup final tkts 21st Ma~       3
## 4        0 U dun say so early hor... U c already then say...                   4
## 5        0 Nah I don't think he goes to usf, he lives around here though       5
## 6        1 FreeMsg Hey there darling it's been 3 week's now and no word~       6
```

We see from the output above that our new `texts` dataframe now has the following fields:

- `category`: 0 if the text is not spam, 1 if it is.
- `text`: the content of the text message
- `text_id`: A unique integer identifier for each text message.

The cell below checks to make sure that both the `category` and `text` fields contain no `NA` or empty values:

```
texts %>%
  filter(
    is.na(category) | is.na(text) | gsub(" ", '', text) == '') %>%
  count()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1     0
```

Now that we know none of our data is missing, we can do some manipulation of the `text` field to make the strings it contains easier for the classifier to make accurate predictions. To do this, the cell below transforms the `text` field into a corpus via the `tm` package. This transformation allows for the usage of the `tm_map` function, which provides a number of handy tools for modifying text used in NLP. In this case, the following changes (along with their justifications) are provided below:

1. **Transform all the text to lowercase:** The classifier we will use words by finding the counts of words that appear in text. We want to make sure that two of the same words are counted together even if they have different capitalization patterns.

2. **Remove all the punctuation from the texts:** Punctuation characters provide no additional useful information for our classifier.

3. **Remove "stop words":**. Stop words are commonly used English words such as "the" and "a", that don't really provide any important information to the text as a whole. By removing these words, the classifier can focus on those words that give more weight to the text's overall meaning. Lists of stop words can vary, but the list used in this case comes from the the `tm` package's `stopwords()` function.

The cell below performs the list transformations listed above, and then adds the cleaned text message back to the `texts` dataframe in a new column called `clean_text`.

```
# transform to corpus
corp <- Corpus(VectorSource(as.list(texts$text)))
# make all text lowercase
corp <- tm_map(corp, content_transformer(tolower))
```

```
## Warning in tm_map.SimpleCorpus(corp, content_transformer(tolower)):
## transformation drops documents
```

```r
# remove all punctuation from text
corp <- tm_map(corp, removePunctuation)
```

```
## Warning in tm_map.SimpleCorpus(corp, removePunctuation): transformation drops
## documents
```

```r
# remove stop words
corp <- tm_map(corp, removeWords, stopwords("en"))
```

```
## Warning in tm_map.SimpleCorpus(corp, removeWords, stopwords("en")):
## transformation drops documents
```

```r
# transform back to dataframe
corp_df <- data.frame(
  text = sapply(corp, as.character), stringsAsFactors = FALSE)
corp_df$text_id <- 1:nrow(corp_df)
colnames(corp_df)[1] <- 'clean_text'

# add cleaned texts back to original dataframe
texts <- texts %>%
  left_join(corp_df, on="text_id")
```

```
## Joining, by = "text_id"
```

```r
texts <- select(texts, text_id, text, clean_text, category)
head(select(texts, text_id, clean_text))
```

```
## # A tibble: 6 x 2
##   text_id clean_text
##     <int> <chr>
## 1       1 go  jurong point crazy available   bugis n great world la e buffet ci~
## 2       2 ok lar joking wif u oni
## 3       3 free entry  2  wkly comp  win fa cup final tkts 21st may 2005 text fa~
## 4       4 u dun say  early hor u c already  say
## 5       5 nah  dont think  goes  usf  lives around  though
## 6       6 freemsg hey  darling   3 weeks now   word back id like  fun     still~
```

Creating a TF-IDF Matrix

To actually train our model, we will use term frequency-inverse document frequency (TF-IDF) matrix. The TD-IDF value for a word in a group of documents (in this case, text messages are our documents) is the result of multiplying together the word's term frequency (information about how much a word appears in the documents) by the word's inverse term frequency (information about the relative rarity of the word in all the documents). This means that a term will be rated as more important if it appears multiple times within the same document, but this importance might wane if the word also appears in numerous other documents. In essence, the TD-IDF metric is just a way of measuring how important each word is in a set of documents. This Medium article provides more in depth explanation, if desired.

In R, creating a TD-IDF matrix is very simple, and is done in the cell below:

```r
tfidf_mat <- DocumentTermMatrix(corp, control = list(weighting = weightTfIdf))

tfidf_df <- as.data.frame(as.matrix(tfidf_mat))
tfidf_df$text_id <- 1:nrow(tfidf_df)

inspect(tfidf_mat)
```

```
## <<DocumentTermMatrix (documents: 5572, terms: 9158)>>
## Non-/sparse entries: 45871/50982505
## Sparsity           : 100%
## Maximal term length: 52
## Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
## Sample             :
##       Terms
## Docs   call can come get ill just lor ltgt now will
##   1445    0   0    0   0   0    0   0    0   0    0
##   1501    0   0    0   0   0    0   0    0   0    0
##   1515    0   0    0   0   0    0   0    0   0    0
##   1525    0   0    0   0   0    0   0    0   0    0
##   1612    0   0    0   0   0    0   0    0   0    0
##   1671    0   0    0   0   0    0   0    0   0    0
##   451     0   0    0   0   0    0   0    0   0    0
##   557     0   0    0   0   0    0   0    0   0    0
##   783     0   0    0   0   0    0   0    0   0    0
##   893     0   0    0   0   0    0   0    0   0    0
```

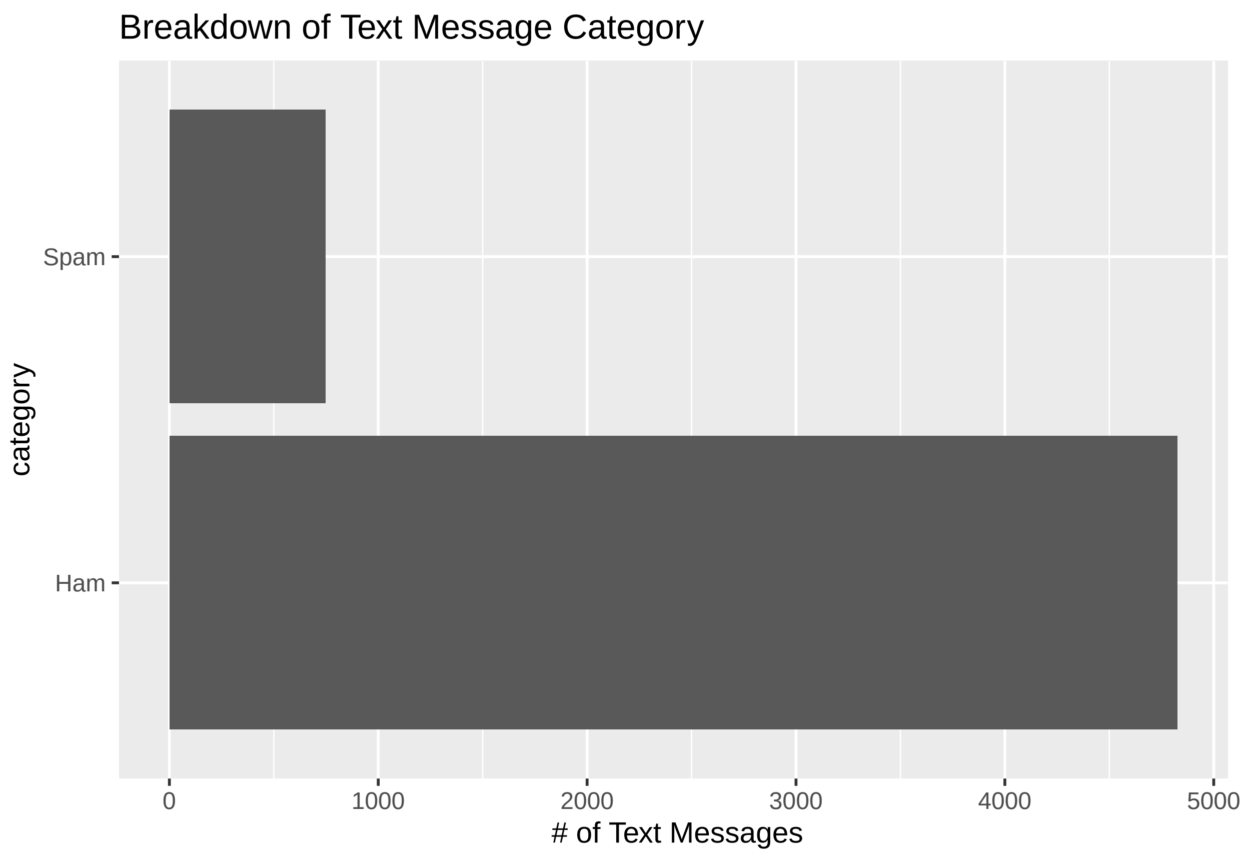
The output above shows what the TF-IDF matrix looks like: it creates a field for every word in the entire set of text messages (9,158) and then a row for each document. The TF-IDF calculation is then performed for each word/document combination and then inputted into the matrix as the values. Note that all the above values are 0, which is to be expected. TD-IDF matrices are typically sparse (contain mostly 0s) due to the fact that each document only contains a small subset of the total words used in the corpus.

Exploratory Data Analysis

Before actually building and training our model, we can used our cleaned dataset and TF-IDF matrix to perform some initial analysis. First, the cell below creates a plot of the breakdown between spam and ham messages:

```r
plt_data <- texts %>%
  group_by(category) %>%
    summarise(count_category = n()) %>%
      mutate(category = ifelse(category==1, 'Spam', 'Ham'))

ggplot(data=plt_data, aes(x=category, y=count_category)) +
  geom_bar(stat='identity') +
  coord_flip() +
  labs(
    y = ' # of Text Messages',
    title = 'Breakdown of Text Message Category'
  )
```



As is probably expected, there are far more legitimate text messages than there are spam ones. The exact numbers are shown below:

```
plt_data
```

```
## # A tibble: 2 x 2
##   category count_category
##   <chr>             <int>
## 1 Ham                4825
## 2 Spam                747
```

The cell below creates a plot of the number of words that occur at each frequency in the corpus.

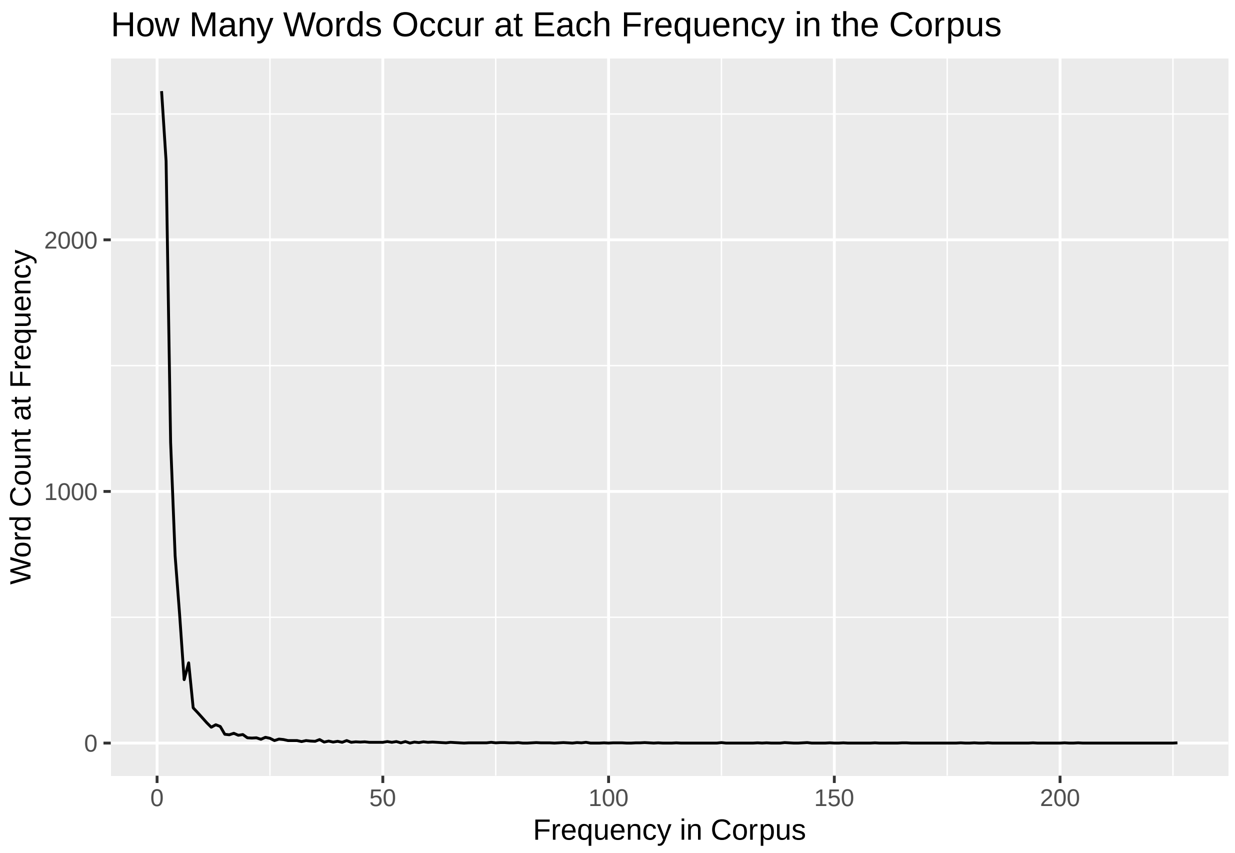
```
num_found_terms = 0
i = 0
freq_counts <- data.frame(matrix(ncol = 2, nrow = 0))


while(num_found_terms < ncol(tfidf_df) - 1) {
  num_words <- length(findFreqTerms(tfidf_mat, i, i+1))
  tmp <- cbind.data.frame(i+1, num_words)
  freq_counts <- rbind(freq_counts, tmp) # (18)

  i <- i + 1
  num_found_terms = num_found_terms + num_words
}

colnames(freq_counts) <- c('frequency', 'num_words')
```

```
ggplot(freq_counts, aes(x=frequency, y=num_words)) +
  geom_line() +
  labs(
    x = "Frequency in Corpus",
    y = "Word Count at Frequency",
    title = "How Many Words Occur at Each Frequency in the Corpus",
  )
```

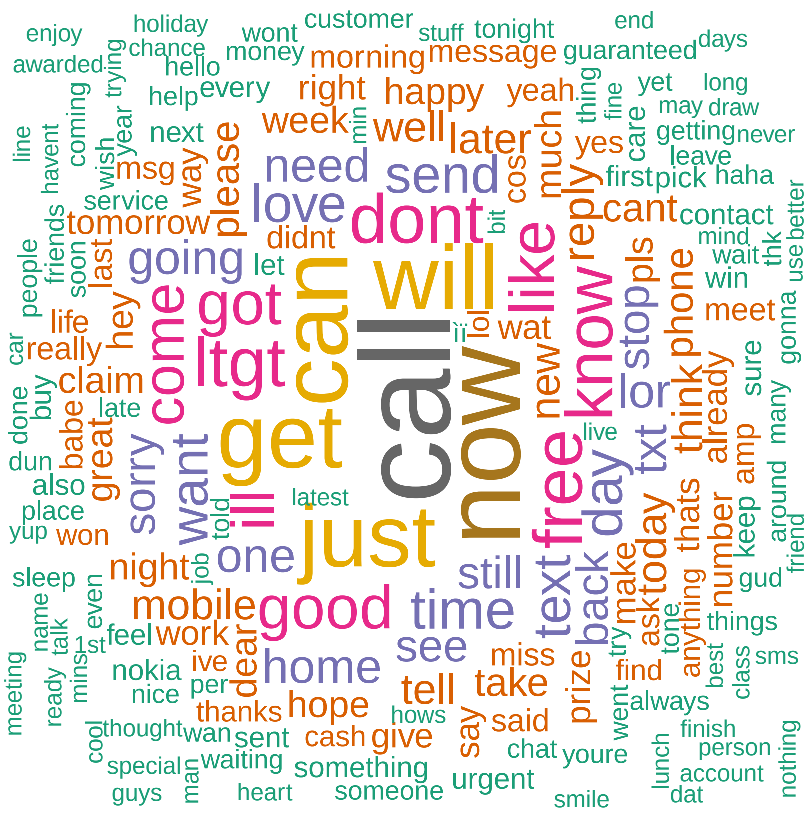


Also as to be expected, the vast majority of words tend to have low frequencies in the corpus. As we move to the right of the plot we see that the words with higher frequencies tend to be part of a smaller group.

Lastly, the cell below gives us an idea of some of the most common words used in our corpus by creating a word cloud:

```
word_cloud_corp <- TermDocumentMatrix(corp)
words <- sort(rowSums(as.matrix(word_cloud_corp)),decreasing=TRUE)
word_cloud_df = df <- data.frame(word = names(words),freq=words)

set.seed(seed=seed)
wordcloud(words = word_cloud_df$word, freq = word_cloud_df$freq, min.freq = 1,           max.words=200,
```



As we can see above, the word cloud indeed shows some very common words you'd expect to see in text messages!

Predict Spam Messages

Prepare Data

Before we begin training our model, the cell below splits our data into testing and training datasets, and stores them as the following four matrices:

- `x_train`: The TF-IDF matrix that will be used to train the Naive Bayes model.
- `y_train`: The category of each text in the `x_train` data.
- `x_test`: The TF-IDF matrix that will be used to create predictions from the trained Naive Bayes model.
- `y_train`: The category of each text in the `y_train` data. It will be used to evaluate the model's performance.

The data was split so that 70% of the data goes to the training dataset.

```r
set.seed(seed = seed)

texts_train <- texts %>% sample_frac(0.70)
texts_test  <- anti_join(texts, texts_train, by = 'text_id')

x_train <- texts_train %>%
  select(text_id) %>%
    inner_join(tfidf_df, by='text_id') %>%
      select(-text_id)
```

```
x_test <- texts_test %>%
  select(text_id) %>%
    inner_join(tfidf_df, by='text_id') %>%
      select(-text_id)

y_train <- as.matrix(select(texts_train, category))
y_train <-
  factor(as.matrix(y_train), levels = c(1, 0), labels = c("yes", "no"))
y_test <- as.matrix(select(texts_test, category))
y_test <-
  factor(as.matrix(y_test), levels = c(1, 0), labels = c("yes", "no"))
```

The cell below creates and trains a Multinomial Naive Bayes model using our training dataset. The Naive Bayes classifier uses Baye's theorem at its core to predict the probabilities that a record belongs to a given category. In this case, the classifier will be used to predict the probability that a text message is spam. If the probability exceeds 50%, we will say that the message has been classified as such. Naive Bayes models are a solid choice for document classification due to this probability calculation, as well as the fact that they can handle large numbers of features without a terrible reduction in performance. The multinomial version of the model was used in this case because the predictor variable values can be continuous, allowing us to use our previously created TF-IDF matrix. More info on how the Naive Bayes classifier works can be found in this medium article.

```
nb_model <- multinomial_naive_bayes(x_train, y_train)
nb_model
```

```
##
## ============================ Multinomial Naive Bayes ============================
##
##  Call:
## multinomial_naive_bayes(x = x_train, y = y_train)
##
## ---------------------------------------------------------------------------------
##
## Laplace smoothing: 0.5
##
## ---------------------------------------------------------------------------------
##
##  A priori probabilities:
##       yes        no
## 0.1371795 0.8628205
##
## ---------------------------------------------------------------------------------
##
##            Classes
## Features             yes           no
##   amore     5.512856e-05 1.599566e-05
##   available 1.715047e-04 3.059707e-04
##   buffet    5.512856e-05 1.599566e-05
##   bugis     5.512856e-05 1.092934e-04
##   cine      5.512856e-05 1.082486e-04
##   crazy     2.098691e-04 3.575294e-04
##   got       1.895890e-04 3.013254e-03
##   great     4.449151e-04 1.873322e-03
##   jurong    5.512856e-05 1.599566e-05
##   point     5.512856e-05 3.211945e-04
##
## ---------------------------------------------------------------------------------
##
## # ... and 9148 more features
##
## ---------------------------------------------------------------------------------
```

Now that the model has been trained, we can use it to predict the class of the texts contained in `x_test`. The cell below creates these predictions and then produces a confusion matrix by comparing these predictions to the actual categories contained in `y_test`.

```
y_pred <- predict(nb_model, as.matrix(x_test), type='class')
confusionMatrix(table(y_pred , y_test))
```

```
## Confusion Matrix and Statistics
##
##       y_test
## y_pred  yes   no
##    yes  206  144
##    no     6 1316
##
##                Accuracy : 0.9103
##                  95% CI : (0.8956, 0.9236)
##     No Information Rate : 0.8732
##     P-Value [Acc > NIR] : 1.147e-06
##
##                   Kappa : 0.683
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9717
##             Specificity : 0.9014
##          Pos Pred Value : 0.5886
##          Neg Pred Value : 0.9955
##              Prevalence : 0.1268
##          Detection Rate : 0.1232
##    Detection Prevalence : 0.2093
##       Balanced Accuracy : 0.9365
##
##        'Positive' Class : yes
##
```

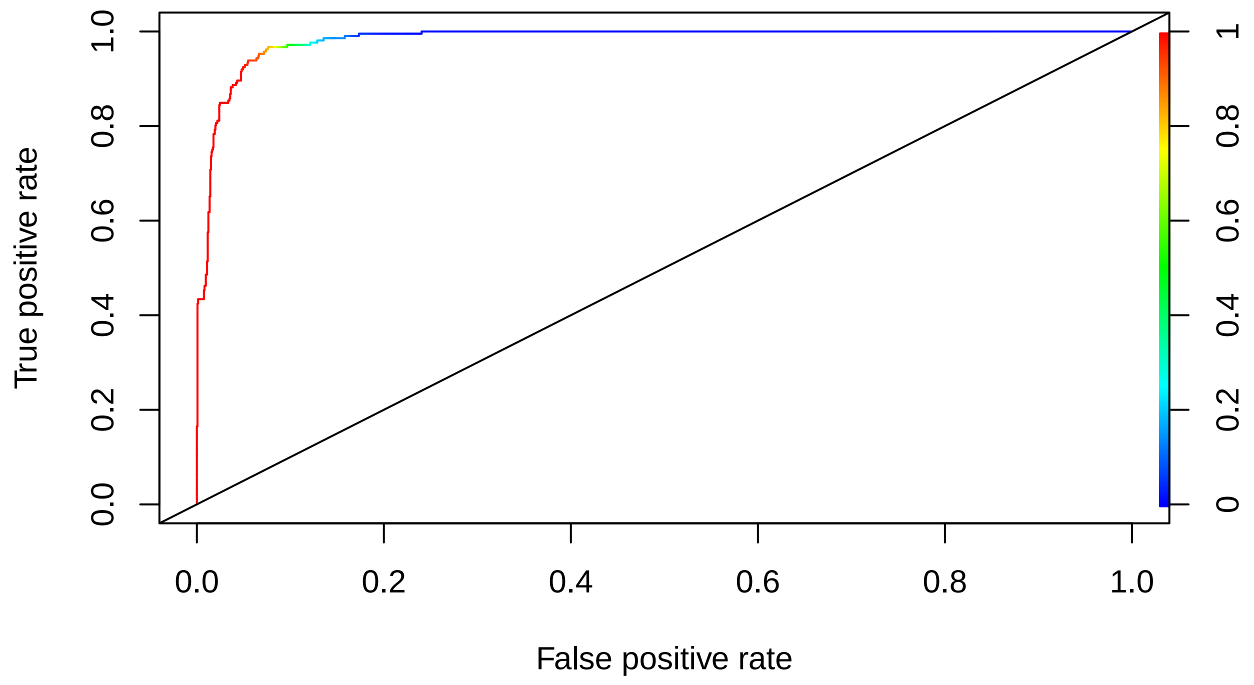
While we see an accuracy score of ~91% (!), we must remember that due to the class imbalance of our dataset it is much more useful to look at metrics such as recall (Sensitivity) and precision (Pos Pred Value). We see from these metrics that while our classifier was able to capture almost every instance of spam (it only missed 6), it was a little overzealous: it incorrectly categorized 144 legitimate text messages as spam.

To get a visual of these metrics we can create a ROC plot, which is done in the cell below.

```
# to get ROC curve we need to first predict probabilities as opposed to class
y_pred_prob <- predict(nb_model, as.matrix(x_test), type='prob')

# create ROC curve
pred = prediction(y_pred_prob[,1], y_test)
roc = performance(pred,"tpr","fpr")
plot(roc, colorize = T)
```

```r
abline(a = 0, b = 1)
```



Given the deviation of the ROC curve from the line $y = x$, we can conclude that our model has performed quite respectably. We can get a quantitative measure of this by calculating the area under this curve (AUC):

```r
auc <- performance(pred, measure = "auc")
auc@y.values[[1]]
```

```
## [1] 0.9830318
```

The high AUC value (it ranges from 0-1) gives us definitive proof of the high quality of the model.

Conclusion

While the Multinomial Naive Bayes model created did show off an impressive ROC curve and AUC score, there is definitely room for improvement. Its weakest aspect by far is its precision: it classified a large number of legitimate text messages as spam. While the recall was extremely high, a better model in this case might be one prioritizes precision over recall. Most people would probably prefer to have a few spam messages sneak in and get all of their legitimate messages, as opposed to having almost no spam but might occasionally miss some important texts. This is a next step for the model, and could be improved by further hyper parameter tuning or the evaluation of different Naive Bayes/word vectorization methodologies.