Spam Filter with Naive Bayes

Jussi Pylkkänen

2025-02-04

Libraries

Here are all of the libraries used in this project:

```r
library(tidyverse)
```

Introduction and preprocess

In this project we will create an algorithm using Naive Bayes Algorithm to detect whether a text message is spam or not. In the end we will have a function which calculates the probability of a message being real and assign a label to it. The function will only work on english messages. This project is based on instructions from Dataquest.io.

Exploring the dataset

Let's open the dataset we'll be using and see what it contains.

```r
spam_original <- read.table(file = "SMSSpamCollection.txt", sep = "\t", header = FALSE,
                            quote = "", stringsAsFactors = FALSE)
colnames(spam_original) <- c("label", "SMS")
spam_original[1:5, 1]
```

```
## [1] "ham"  "ham"  "spam" "ham"  "ham"
```

```r
spam_original[1:5, 2]
```

```
## [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there g
## [2] "Ok lar... Joking wif u oni..."
## [3] "Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive
## [4] "U dun say so early hor... U c already then say..."
## [5] "Nah I don't think he goes to usf, he lives around here though"
```

```r
spam_original$label <- as.factor(spam_original$label)
total_rows <- nrow(spam_original); total_rows
```

```
## [1] 5574
```

```r
ncol(spam_original) # Total no. columns
```

```
## [1] 2
```

```r
round(prop.table(table(spam_original$label)) * 100, 2) #Percentage of ham and spam
```

```
##
##  ham spam
## 86.6 13.4
```

First, we can see what the labels and texts look like. We can see that the dataset contains 5574 observations, each with 2 variables. The first variable is a factor of two levels, "spam" or "ham" (meaning not-spam). The second variable is an SMS message which has been flagged as either spam or not by a human. The percentage of spam is 13.4%

Dividing the dataset

Let's set 80 % of the data to be the training set, and the remaining 20 % will be used for testing. We want to take them randomly to reduce possible bias so let's use sample() function.

```r
train_index <- sample(1:nrow(spam_original), 0.8 * nrow(spam_original))

train_set <- spam_original[train_index, ]
test_set <- spam_original[-train_index, ]

round(prop.table(table(train_set$label)) * 100, 2) #Percentage of ham and spam
```

```
##
##   ham  spam
## 86.45 13.55
```

```r
round(prop.table(table(test_set$label)) * 100, 2)
```

```
##
##   ham  spam
## 87.17 12.83
```

Cleaning the data

Next, let's convert all the letters to lower case, so that for example "CAR" and "car" are treated as the same word.

```r
train_clean <- spam_original %>%
  mutate(
    SMS = SMS %>%
      str_to_lower() %>%    # Convert to lowercase
      str_squish()      # Remove empty spaces
    )
```

Calculations

Now that we have preprocesses our dataset, we can move onto calculating the probabilities. In a Naive Bayes algorithm, we calculate the probability of the message being labeled as either 'spam' or 'ham'. This is done by multiplying the prior probability of the label with the product of the probabilities of each word given the label, under the assumption that the words are conditionally independent given the label.

Frequencies

First we need to calculate all the frequencies of each word for both spam messages and ham messages. Then we input those into a single tibble, so we can later easily calculate the needed probabilities.

```r
# Calculate the amount of all spam messages, all ham messages
N_spam <- sum(spam_original$label == "spam")
N_ham <- sum(spam_original$label == "ham")

# Split the train set and calculate P(Spam) and P(Ham)
spam_train <- filter(train_clean, label == "spam")
P_spam <- nrow(spam_train) / nrow(spam_original)

ham_train <- filter(train_clean, label == "ham")
P_ham <- nrow(ham_train) / nrow(spam_original)

# Isolate the Spam and Ham messages into two tibbles and split the messages into words
spam_tibble <- tibble(SMS = spam_train$SMS)
spam_words <- spam_tibble$SMS %>%
  str_split("\\s+") %>%
  unlist() %>%
  unique()

ham_tibble <- tibble(SMS = ham_train$SMS)
ham_words <- ham_tibble$SMS %>%
  str_split("\\s+") %>%
  unlist() %>%
  unique()

# Create our vocabulary
vocabulary <- unique(c(spam_words, ham_words))
N_vocabulary <- length(vocabulary)

# Count the occurrences of a word in a set of messages
count_word_occurrences <- function(message_set, word){
  sum(str_count(message_set, fixed(word)))
}

# Create a tibble with three columns: Word, occurrences in spam and occurrences in ham
word_frequencies <- tibble(
  word = vocabulary,
  spam_count = sapply(vocabulary, count_word_occurrences, message_set = spam_tibble$SMS),
  ham_count = sapply(vocabulary, count_word_occurrences, message_set = ham_tibble$SMS)
)
```

Classification Function

Now that we have our tibble of frequencies, we can finally move onto the actual classification. First we need to clean the message as before and splice it into individual words.

Then we calculate the probability of each word given the label. In short we need to see how many times a word appears in the set and divide that with the amount of words in said set plus all the words in the dictionary. Then we multiply the prior with each of these probabilities to get the probability of label given the message.

Then we simply compare which labels has higher final probability and assign the message to that label.

```r
classify_message <- function(message) {
  clean_message <- message %>%
    str_to_lower() %>%
    str_squish()

  message_words <- str_split(clean_message, "\\s+") %>% unlist()

  # Prior probabilities
  P_spam_given_message <- P_spam
  P_ham_given_message <- P_ham

  # Iterate over each word in the message. Ignore words not in the vocabulary
  for (word in message_words) {
    if (word %in% vocabulary) {
      # Get the frequencies of the word in spam and ham
      spam_count <- word_frequencies$spam_count[word_frequencies$word == word]
      ham_count <- word_frequencies$ham_count[word_frequencies$word == word]

      # If the word is not found in the word_frequencies, set counts to 0
      if (length(spam_count) == 0) spam_count <- 0
      if (length(ham_count) == 0) ham_count <- 0

      # Probability of word given label
      P_word_given_spam <- (spam_count) / (N_spam * N_vocabulary)
      P_word_given_ham <- (ham_count) / (N_ham * N_vocabulary)

      # Multiply the probabilities
      P_spam_given_message <- P_spam_given_message * P_word_given_spam
      P_ham_given_message <- P_ham_given_message * P_word_given_ham
    }
  }

  if (!is.na(P_spam_given_message) && !is.na(P_ham_given_message)) {
    if (P_spam_given_message > P_ham_given_message) {
      return("spam")
    } else {
      return("ham")
    }
  } else {
    return("ham")  # Fallback to "ham" in case of missing data
  }
}
```

Now that we have out classification function, we need to test the accuracy.

```r
train_set <- train_set %>%
  mutate(predicted_label = map_chr(SMS, classify_message))

accuracy <- round(mean(train_set$predicted_label == train_set$label), 3) * 100; accuracy
```

```
## [1] 99.3
```

The accuracy is very high. That is to be expected since it's the training set. Let's see how good it is for the test set.

```r
test_set <- test_set %>%
  mutate(predicted_label = map_chr(SMS, classify_message))

accuracy <- round(mean(test_set$predicted_label == test_set$label), 3) * 100; accuracy
```

```
## [1] 99.1
```

We can see that the function performs very well even for the test set.

Closing words

In conclusion, we successfully developed a Spam filter using Naive Bayes algorithm. The model could still be tweaked with for example some smoothing parameters but this is good for now. After experimenting with different preprocessing methods, I found out that the raw text produced the best accuracy of 99.1%. This project has provided me with a very good insight on Bayesian Theorem and classification tasks and also has demonstrated how a simple approach can yield strong results.