Building a spam filter with a Naive Bayes classifier

Emmanuel Messori

24/09/2021

Objective

We want to build an algorithm for spam detection using a Naive Bayes classifier.

Theoretical introduction

To be able to build a classifier which automatically classifies the messages as "spam" or "ham", we are interested in determining the probability of a message being spam given its composition and also the probability of it being ham (not spam):

$P(\text{Spam}|w_1, w_2, w_3, ..., w_n)$

$P(\text{Spam}^c|w_1, w_2, w_3, ..., w_n)$

Using Bayes theorem we obtain this formula:

$P(\text{Spam}|w_1, w_2, w_3, ..., w_n) = \frac{P(w_1,w_2,w_3,...|Spam) \times P(Spam)}{P(w_1,w_2,w_3,...)}$

Since we are dealing with the Naive Bayes algorithm, we can safely ignore the denominator and use the numerator to perform the classification:

$P(\text{Spam}|w_1, w_2, w_3, w_4) \propto P(\text{Spam}) \times P(w_1, w_2, w_3, w_4|\text{Spam})$

We want to build a classifier based on the Naive Bayes algorithm which supposes "naively" independency between the event at hand, with two assumptions:

1. We'll model a multi-word message as an intersection of many single words. In mathematical notation, we would rewrite the conditional probability in as:

$P(w_1, w_2, w_3, w_4|\text{Spam}) = P(w_1 \cap w_2 \cap w_3 \cap w_4|\text{Spam})$

2. Each of the words in the messages are conditionally independent. Conditional independence is essentially the same as regular independence, where the only difference between the two is that the two events are independent conditioned on another one. We would write conditional independence mathematically as:

$P(A \cap B|C) = P(A|C) \times P(B|C)$

From this assumptions, we can derive this formula to calculate the conditional probability:

$P(\text{Spam}|w_1, w_2, \ldots, w_n) \propto P(\text{Spam}) \times \prod_{i=1}^{n} P(w_i|\text{Spam})$

Meaning that the conditional probability of a message being spam given certain words is *proportional* to the product of the probability of a spam message times the probability of occurrence of every word in a spam message.

We'll use addictive smoothing to avoid 0 values in the products(when a word occurs only in one message or either in the spam or non spam messages).The smoothing parameter will prevent the numerator from being zero, without changing the original probability too much. However, if we want to introduce additive smoothing, we have to add it too all of the word probabilities, not just the words that are absent from our vocabulary. In more general terms, this is the equation that we'll need to use for every word probability:

$P(w|\text{Spam}) = \frac{N_{w|\text{Spam}}+\alpha}{N_{\text{Spam}}+\alpha \times N_{Vocabulary}}$

Where N_w is the number of occurrences of a single word and Vocabulary represents all the unique words used in the messages. When $\alpha = 1$ the additive smoothing technique is most commonly known as Laplace smoothing (or add-one smoothing). Here we'll test different values to find the optimal one.

The Data

Our first task is to provide a computer with the information on how to classify messages. To do that, we'll use the Naive Bayes algorithm on a dataset of 5,572 SMS messages that have already been classified by humans.

The dataset was put together by Tiago A. Almeida and José María Gómez Hidalgo, and it can be downloaded from the The UCI Machine Learning Repository. The data collection process is described in more details on this page, where you can also find some of the authors' papers.

Note that due to the nature of spam messages, the dataset contains content that may be offensive to some users.

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.4     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.1     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
# spam_reduced <- read_csv('https://dq-content.s3.amazonaws.com/475/spam.csv')
spam <- read_delim("SMSSpamCollection", col_names = c("label", "sms"))
```

```
## Rows: 4773 Columns: 2

## -- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr (2): label, sms

##
## 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.
```

```r
head(spam)
```

```
## # A tibble: 6 x 2
##   label sms
##   <chr> <chr>
## 1 ham   Go until jurong point, crazy.. Available only in bugis n great world la~
## 2 ham   Ok lar... Joking wif u oni...
## 3 spam  Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Tex~
## 4 ham   U dun say so early hor... U c already then say...
## 5 ham   Nah I don't think he goes to usf, he lives around here though
## 6 spam  FreeMsg Hey there darling it's been 3 week's now and no word back! I'd ~
```

```r
spam %>%
    count(label) %>%
    ungroup() %>%
    mutate(prop = n/sum(n) * 100)
```

```
## # A tibble: 2 x 3
##   label     n  prop
##   <chr> <int> <dbl>
## 1 ham    4123  86.4
## 2 spam    650  13.6
```

86% of the messages are definitely normal messages, when 14% are classified as spam.

Wordcloud

Before starting to build our classifier, we will visualize the words in the dataset, excluding stopwords.

```r
library(tm)
```

```
## Le chargement a nécessité le package : NLP

##
## Attachement du package : 'NLP'

## L'objet suivant est masqué depuis 'package:ggplot2':
##
##     annotate
```

```r
library(wordcloud)
```

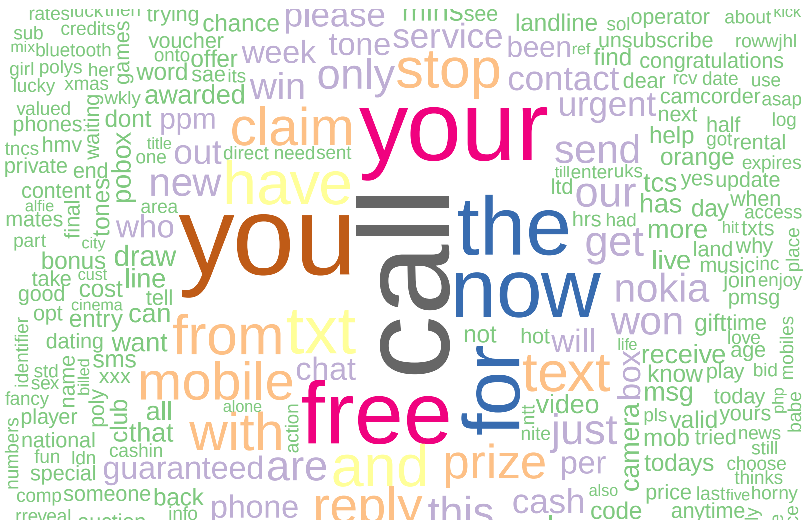
```
## Le chargement a nécessité le package : RColorBrewer
```

```r
library(RColorBrewer)


plotcloud <- function(df, contr = FALSE, dash = FALSE, palette = "Accent") {
    sms <- df$sms
```

```
    corp <- VCorpus(VectorSource(sms))
    corp <- tm_map(corp, content_transformer(tolower))
    corp <- tm_map(corp, removeNumbers)
    corp <- tm_map(corp, removePunctuation, preserve_intra_word_contractions = contr,
        preserve_intra_word_dashes = dash)
    corp <- tm_map(corp, stripWhitespace)
    wordcloud(corp, random.order = FALSE, colors = RColorBrewer::brewer.pal(8, palette))
    invisible(corp)
}

corpuses <- spam %>%
    split(., ~label) %>%
    map(plotcloud)
```



Term frequencies

```r
hamdtm <- DocumentTermMatrix(corpuses$ham)
spamdtm <- DocumentTermMatrix(corpuses$spam)
head(sort(colSums(as.matrix(hamdtm)), decreasing = TRUE), 10)
```

```
##  you  the  and  for that have your  but  not  are
## 1705 1038  784  480  458  409  403  393  379  376
```

```r
head(sort(colSums(as.matrix(spamdtm)), decreasing = TRUE), 10)
```

```
## call  you your free  now  the  for  txt have  and
##  294  251  215  191  176  176  169  133  116  114
```

```r
# only one long message is the source of all the 'ham' in the hams
head(sort(termFreq(corpuses$ham[[3941]]$content), decreasing = TRUE))
```

```
##  ham  you  the  and your spam
##  269   78   63   47   46   39
```

A classifier using the naivebayes package

```r
wholecorpus <- VCorpus(VectorSource(spam$sms))
wholecorpus <- tm_map(wholecorpus, stripWhitespace)
wholecorpus <- tm_map(wholecorpus, content_transformer(tolower))
wholecorpus <- tm_map(wholecorpus, removePunctuation)
wholedtm <- DocumentTermMatrix(wholecorpus)

# term frequencies
head(sort(colSums(as.matrix(wholedtm)), decreasing = TRUE))
```

```
##  you  the  and  for your call
## 1956 1213  898  649  618  525
```

```r
naivebayes::multinomial_naive_bayes(as.matrix(wholedtm), y = spam$label) -> clf
clf
```

```
##
## ============================ Multinomial Naive Bayes ============================
##
##  Call:
## naivebayes::multinomial_naive_bayes(x = as.matrix(wholedtm),
##     y = spam$label)
##
## ---------------------------------------------------------------------------------
##
## Laplace smoothing: 0.5
##
## ---------------------------------------------------------------------------------
##
##  A priori probabilities:
##       ham      spam
## 0.8638173 0.1361827
##
## ---------------------------------------------------------------------------------
##
##            Classes
## Features             ham         spam
##   "harry    9.845136e-06 8.975318e-05
##   £10       9.845136e-06 3.290950e-04
##   £100      2.953541e-05 1.226627e-03
##   £1000     4.922568e-05 1.824981e-03
##   £10000    2.953541e-05 2.692595e-04
##   £100000   9.845136e-06 8.975318e-05
##   £1000call 2.953541e-05 2.991773e-05
##   £12       9.845136e-06 8.975318e-05
##   £125      9.845136e-06 8.975318e-05
##   £1250     9.845136e-06 1.495886e-04
##
## ---------------------------------------------------------------------------------
##
## # ... and 8877 more features
```

```
##
## ----------------------------------------------------------------------------------
```

```r
predict(clf, newdata = as.matrix(DocumentTermMatrix(Corpus(VectorSource(c(sms = wholecorpus[[6]]))))))
```

```
## [1] spam
## Levels: ham spam
```

Building the model

We want to optimize our algorithm's ability to correctly classify messages that it hasn't seen before. We'll want to create a process by which we can tweak aspects of our algorithm to see what produces the best predictions. The first step we need to take towards this process is divide up our spam data into 3 distinct datasets.

- A training set, which we'll use to "train" the computer how to classify messages.
- A cross-validation set, which we'll use to assess how different choices of α affect the prediction accuracy.
- A test set, which we'll use to test how good the spam filter is with classifying new messages. We're going to keep 80% of our dataset for training, 10% for cross-validation and 10% for testing.

We expose the algorithm to examples of spam and ham through the training set. In other words, we develop all of the conditional probabilities and vocabulary from the training set. After this, we need to choose an α value. The cross-validation set will help us choose the best one. Throughout this whole process, we have a set of data that the algorithm never sees: the test set. We hope to maximize the prediction accuracy in the cross-validation set since it is a proxy for how well it will perform in the test set.

```r
set.seed(1)
trindex <- sample(nrow(spam), size = 0.8 * nrow(spam))

# train and test sets

train <- spam[trindex, ]

testset <- spam[-trindex, ]

# divide the test into cv and test

set.seed(1)
testindex <- sample(nrow(testset), size = 0.5 * nrow(testset))

cv <- testset[testindex, ]
test <- testset[-testindex, ]
```

Using the tm library to obtain a dictionary with word frequencies

We should create a vocabulary of all the words used in the messages. To do that, we have to clean the data convert it into a format that makes it easier to get the information we need :

- all the messages should be converted to lowercase
- whitespaces and punctuation should be removed

- finally we want to split the sentences into single words and calculate their frequencies

To do that, the easiest way is to use the `tm` package.

```
# map approach to obtain two different spam and ham tibbles
dfs <- train %>%
    split(~label) %>%
    # create a Corpus object
map(~VCorpus(VectorSource(.$sms))) %>%
    # transform to lowercase
map(~tm_map(., content_transformer(tolower))) %>%
    # remove numbers
map(~tm_map(., removeNumbers)) %>%
    # remove punctuation
map(~tm_map(., removePunctuation)) %>%
    # ..and strip whitespace
map(~tm_map(., stripWhitespace)) %>%
    # obtain a dataframe of word frequencies
map(~data.frame(freq = colSums(as.matrix(DocumentTermMatrix(.)))))

trspam <- dfs$spam %>%
    rownames_to_column("word")
trham <- dfs$ham %>%
    rownames_to_column("word")

voc <- full_join(trham, trspam, by = "word", suffix = c(".ham", ".spam")) %>%
    replace_na(list(freq.ham = 0, freq.spam = 0))

slice_max(voc, n = 10, order_by = freq.spam)
```

```
##    word freq.ham freq.spam
## 1  call      171       229
## 2   you     1289       206
## 3  your      286       169
## 4  free       45       149
## 5   the      788       132
## 6   for      374       128
## 7   now      201       128
## 8   txt       12        90
## 9  have      315        88
## 10 from       99        86
```

Base classifier

Now that we're done with data cleaning and the vocabulary, we can start calculating the probabilities needed to start classification.

```
# total number of spam words
Nspam <- sum(voc$freq.spam)
# total number of ham words
Nham <- sum(voc$freq.ham)
# length of the vocabulary
```

```r
Nvocabulary <- nrow(voc)
# prior probability of spam
Pspam <- nrow(filter(train, label == "spam"))/nrow(train)
# prior probability of ham
Pham <- 1 - Pspam
```

```r
sentence <- "You won 10000000 $! Claim your prize!"

classifier <- function(message=sentence, alpha=1) {

  message%>%                                          #cleaning the function input
    str_to_lower() %>%
    str_remove_all('[:digit:]') %>%
    str_remove_all('[:punct:]') %>%
    str_squish() %>% str_split(" ")%>%
    unlist() -> words
probs_spam <- numeric()
probs_ham <- numeric()
for (w in words){
  if (w %in% voc$word) {
freq.ham <- filter(voc, word == w) %>% pull(freq.ham)
freq.spam <- filter(voc, word == w) %>% pull(freq.spam)
probs_spam[w] <- (freq.spam + alpha) / (Nspam + (alpha * Nvocabulary))   #formula for smoothed conditio
probs_ham[w] <- (freq.ham + alpha) / (Nspam + (alpha * Nvocabulary))
}
}
p_spam_given_words <- prod(probs_spam) * Pspam
p_ham_given_words <- prod(probs_ham) * Pham

#invisible(c(p_ham_given_words, p_spam_given_words))
ifelse(p_spam_given_words >= p_ham_given_words, "spam", "ham")
}

classifier()
```

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

We can now apply the function to the train dataset to see how well it performs:

```r
train %>%
    mutate(pred = map_chr(sms, classifier)) -> train
(cm <- table(train$pred, train$label))
```

```
##
##          ham spam
##   ham   3318  172
##   spam     0  328
```

```r
acc <- (cm[1] + cm[4])/sum(cm)
```

The accuracy of the classifier is 0.95%. We can probably improve the model with hyper parameter tuning.

Alpha tuning

What we need to do now is to repeat this process multiple times using different values of α. For each α we'll assess the classifier's accuracy on the cross-validation set. We'll choose the α that maximizes the prediction accuracy in the cross-validation set.

```
alpha <- seq(0.001, 1, length.out = 5)
accuracy <- numeric()
for (a in alpha) {
    cv %>%
        mutate(pred = map_chr(sms, classifier, alpha = a)) -> cv
    cm <- table(cv$pred, cv$label)
    acc <- (cm[1] + cm[4])/sum(cm)
    accuracy <- c(accuracy, acc)
}
(results <- tibble(alpha = alpha, acc = accuracy))
```

```
## # A tibble: 5 x 2
##   alpha   acc
##   <dbl> <dbl>
## 1 0.001 0.971
## 2 0.251 0.958
## 3 0.500 0.956
## 4 0.750 0.952
## 5 1     0.950
```

Low values of alpha give the best accuracy.

Test accuracy

For the purpose of the exercise we will retain $\alpha = 0.001$.

```
test %>%
    mutate(pred = map_chr(sms, classifier, alpha = 0.001)) -> test
(cm <- table(test$pred, test$label))
```

```
##
##         ham spam
##   ham   393   21
##   spam    0   64
```

```
(acc <- (cm[1] + cm[4])/sum(cm))
```

```
## [1] 0.9560669
```

We obtain 95% accuracy on the test set. Only 21 "spam" messages are misclassified as "ham" (false negatives).