Project 4

Alice Ding

2023-04-28

Overview

```
It can be useful to be able to classify new "test" documents using already classified
"training" documents.  A common example is using a corpus of labeled spam and ham (non-spam)
e-mails to predict whether or not a new document is spam.
```

For this project, I'll be taking a list of 2,551 ham (non-spam) messages and 1,398 spam messages to see whether a document is spam or not.

Load Data

To start, we'll download the files and unzip them in a `project4` folder.

```r
# download the ham file
download.file(url = "http://spamassassin.apache.org/old/publiccorpus/20021010_easy_ham.tar.bz2"
              , destfile = "20021010_easy_ham.tar.bz2")

# extract/unzip file
untar("20021010_easy_ham.tar.bz2"
      , exdir = "project4"
      , compressed = "bzip2")

# download the spam file
download.file(url = "https://spamassassin.apache.org/old/publiccorpus/20030228_spam_2.tar.bz2"
              , destfile = "10030228_spam_2.tar.bz2")

untar("10030228_spam_2.tar.bz2"
      , exdir="project4"
      , compressed = "bzip2")
```

Data Preparation

Now that the files are downloaded, they're located in two folders: `easy_ham` and `spam_2`. We'll use a function, `create_df` to extract the information from these files and load them into two dataframes, then combine

```r
library(tidyverse)
library(tm)
library(tidytext)
library(dplyr)
library(randomForest)
```

```r
library(caret)

ham <- './project4/easy_ham'
spam <- './project4/spam_2'

create_df <- function(path, tag){
  files <- list.files(path = path
                      , full.names = TRUE
                      , recursive = TRUE)
  email <- lapply(files, function(x) {
    body <- read_file(x)
    }
  )
  email <- unlist(email)
  data <- as.data.frame(email)
  data$tag <- tag
  return(data)
}

ham_df <- create_df(ham, tag="ham")
spam_df <- create_df(spam, tag="spam")
full_df <- rbind(ham_df, spam_df)
table(full_df$tag)
```

```
##
##  ham spam
## 2551 1398
```

We've successfully imported all the emails and tagged them! Now, we'll want to clean the `email` column to only hold information we need.

```r
full_df <- full_df |>
  mutate(email = str_remove_all(email, pattern = "<.*?>")) |>
  mutate(email = str_remove_all(email, pattern = "[:digit:]")) |>
  mutate(email = str_remove_all(email, pattern = "[:punct:]")) |>
  # remove new lines
  mutate(email = str_remove_all(email, pattern = "[\n]")) |>
  # lowercase everything
  mutate(email = str_to_lower(email)) |>
  unnest_tokens(output = text, input = email,
                token = "paragraphs",
                format = "text") |>
  anti_join(stop_words, by=c("text" = "word"))
```

Looks much cleaner!

Next, we'll transform the words in the `text` column into a corpus of messages using the `tm_map` function. We'll also remove whitespace, numbers, stop words, etc. in this process.

```r
corp <- VCorpus(VectorSource(full_df$text))
corp <- tm_map(corp, removeNumbers)
corp <- tm_map(corp, removePunctuation)
corp <- tm_map(corp, removeWords, stopwords("english"))
```

```
corp <- tm_map(corp, content_transformer(stringi::stri_trans_tolower))
corp <- tm_map(corp, stripWhitespace)
corp <- tm_map(corp, stemDocument)
```

Now, using the `DocumentTermMatrix()` function, we will create a document Term Matrix from the dataframe and remove sparse terms with `removeSparseTerms()`. After, we will convert it back to a dataframe and mark emails with 0 and 1 for ham and spam respectively. This will leave us with a dataframe that has a column per word/term, whether that word exists in the email text, and the final classification of spam vs. ham.

```
d <- removeSparseTerms(DocumentTermMatrix(corp, control = list(stemming = TRUE)), 0.999)

convert_count <- function(x) {
  result <- ifelse(x > 0, 1, 0)
  result <- factor(result, levels = c(0, 1), labels = c(0, 1))
  result
}

temp <- apply(d, 2, convert_count)

full_df_matrix <- as.data.frame(as.matrix(temp))

full_df_matrix$class <- full_df_matrix$class
```

Predicting

We'll use 70% of the data as training data and 30% for testing. For the model, we'll try the `randomForest` classifier with 300 trees.

```
set.seed(1234)
prediction <- createDataPartition(full_df_matrix$class, p = 0.7, list = FALSE, times = 1)

# create the training and testing dataset
training <- full_df[prediction,]
# testing is just everything not in the training set
testing <- full_df[-prediction,]

# have to turn these into factors, otherwise the randomForest thing won't work
training$tag <- factor(training$tag)
testing$tag <- factor(testing$tag)

classifier <- randomForest(x = training, y = training$tag, ntree = 300)
predicted <- predict(classifier, newdata = testing)
confusionMatrix(table(predicted, testing$tag))
```

```
## Confusion Matrix and Statistics
##
##
## predicted ham spam
##      ham  774    0
##      spam   0  410
```

```
##
##                Accuracy : 1
##                  95% CI : (0.9969, 1)
##     No Information Rate : 0.6537
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.6537
##          Detection Rate : 0.6537
##    Detection Prevalence : 0.6537
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : ham
##
```

We can see a 99.69% accuracy for this model – pretty good!

Conclusion

Spam detection ultimately takes a lot of effort to get right. From acquiring and cleaning the data to then preparing it for analysis, the task of actually running the model arguably took the least amount of effort in terms of research done to implement it. Using `randomForest`, the model was pretty successful; a next step would be to use various other models (Naive Bayes as an example) to compare and see which is most efficient.