# Proj_4

Mathew Katz

2022-11-19

Project 4:

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, you can start with a spam/ham dataset, then predict the class of new documents (either withheld from the training dataset or from another source such as your own spam folder). One example corpus: https://spamassassin.apache.org/old/publiccorpus/

I used two files from the above website; 1. 20030228_spam_2.tar.bz2 2. 20021010_easy_ham.tar.bz2

Imports:

Importing of the Spam and Ham files:

```
spam_files <- dir('spam_2/')
ham_files <- dir('easy_ham/')

#Clean spam files
spam_emails <- c()
for(i in 1:length(spam_files)) {
  file <- paste0('spam_2/', spam_files[i])
  con <- file(file, open="rb", encoding="latin1")
  txt <- readLines(con)
  msg <- txt[seq(which(txt=="")[1]+1, length(txt), 1)]
  close(con)
  email <- c(i,paste(msg, collapse=" "))
  spam_emails <- rbind(spam_emails, email)
}

#Turn spam file into a dataframe with 1 representing a spam column
spam <- data.frame(spam_emails, stringsAsFactors=FALSE, row.names=NULL)
names(spam) <- c('num', 'txt')
spam <- mutate(spam, spam = 1)

#Clean ham files
ham_emails <- c()
for(i in 1:length(ham_files)) {
  file <- paste0('easy_ham/', ham_files[i])
  con <- file(file, open="rb", encoding="latin1")
  txt <- readLines(con)
  msg <- txt[seq(which(txt=="")[1]+1, length(txt), 1)]
  close(con)
```

```r
  email <- c(i,paste(msg, collapse=" "))
  ham_emails <- rbind(ham_emails, email)
}
#Turn ham file into a dataframe with 0 representing a ham column
ham <- data.frame(ham_emails, stringsAsFactors=FALSE, row.names=NULL)
names(ham) <- c('num', 'txt')
ham <- mutate(ham, spam = 0)

#Combine ham and spam dataframes
set.seed(6241998)
spam_ham <- bind_rows(spam, ham)
spam_ham$spam <- as.character(spam_ham$spam)
spam_ham <- spam_ham[sample(nrow(spam_ham)),]
```

Tokenization (Take raw data and converts it into a useful data string. This is often used in natural language processing to split paragraphs and sentences into smaller units that can be more easily assigned meaning.):

```r
tokens <- spam_ham %>%
    unnest_tokens(output = word, input = txt) %>%
    filter(!str_detect(word, '^[[:digit:]]*$')) %>%
    filter(!str_detect(word, '\\B[[:digit:]]*$')) %>%
    filter(!str_detect(word, '^*[[:digit:]]')) %>%
    filter(!str_detect(word, '<(.+?)>')) %>%
    filter(!str_detect(word, '^[[:punct:]]*$')) %>%
    filter(!str_detect(word, '\\B[[:punct:]]*$')) %>%
    filter(!str_detect(word, '^*[[:punct:]]')) %>%
    anti_join(stop_words) %>%
    mutate(word = wordStem(word))
```

```
## Joining, by = "word"
```

Taking a look at the tokens:

```r
head(tokens)
```

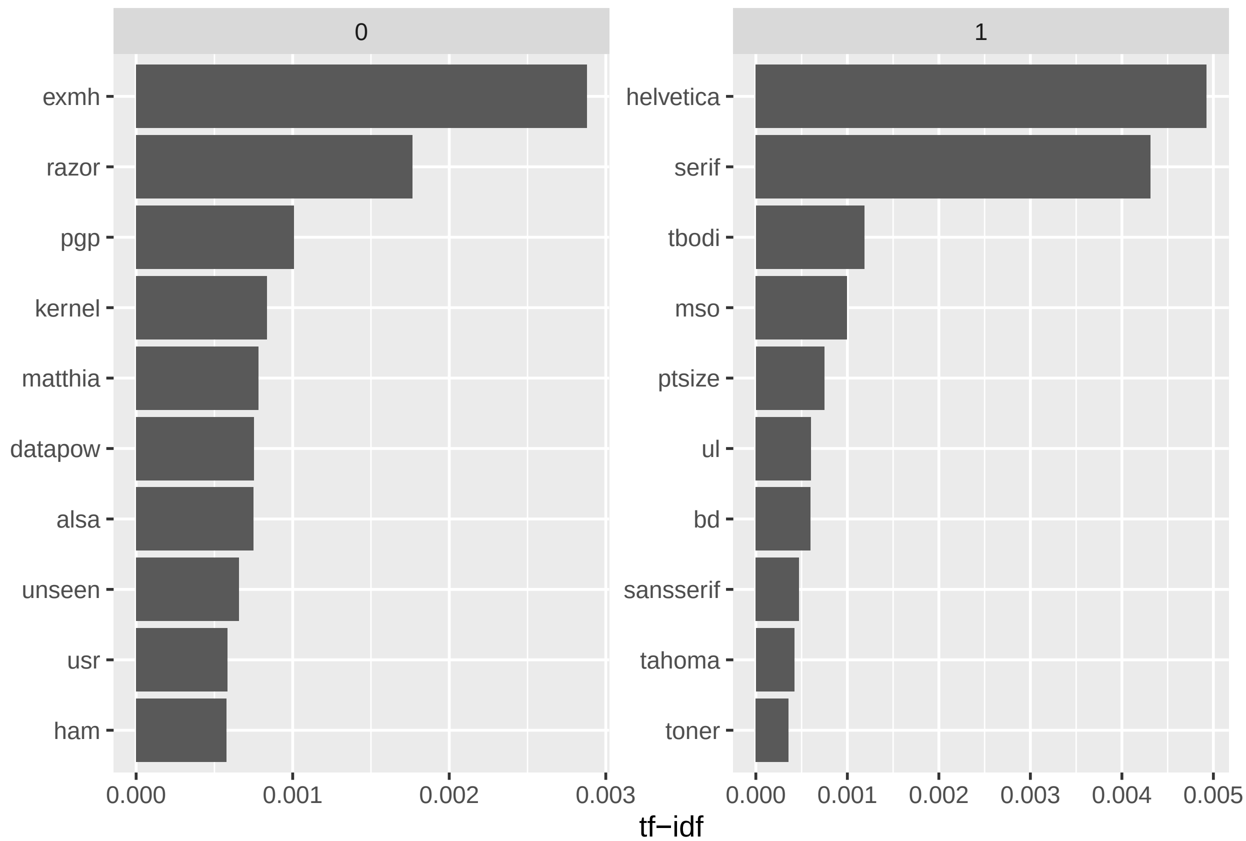
```
##    num spam    word
## 1 1282    0 matthia
## 2 1282    0     mon
## 3 1282    0     sep
## 4 1282    0 matthia
## 5 1282    0    saou
## 6 1282    0   wrote
```

Top 10 tokens among ham (0) and spam (1) emails:

```r
dtm_tfidf <- tokens %>%
  count(spam, word) %>%
  bind_tf_idf(term = word, document = spam, n = n)
dtm_plot <- dtm_tfidf %>%
  arrange(desc(tf_idf)) %>%
  mutate(word = factor(word, levels = rev(unique(word))))
```

```
dtm_plot %>%
  group_by(spam) %>%
  top_n(10) %>%
  ungroup() %>%
  mutate(word = reorder_within(word, tf_idf, spam)) %>%
  ggplot(aes(word, tf_idf)) +
  geom_col() +
  scale_x_reordered() +
  labs(x = NULL, y = "tf-idf") +
  facet_wrap(~ spam, scales = "free") +
  coord_flip()
```

```
## Selecting by tf_idf
```




Tokens to Document Term Matrix (mathematical matrix that describes the frequency of terms that occur in a collection of documents [rows correspond to documents in the collection and columns correspond to terms]):

```
dtm <- tokens %>%
   count(num, word) %>%
   cast_dtm(document = num, term = word, value = n)
dtm
```

```
## <<DocumentTermMatrix (documents: 2551, terms: 25322)>>
## Non-/sparse entries: 294454/64301968
```

```
## Sparsity           : 100%
## Maximal term length: 121
## Weighting          : term frequency (tf)
```

Remove tokens that appear in less than or equal to 1% of all documents in the matrix. This reduces the number of tokens from 25,322 to 2,061 and the maximal term length from 121 characters to 33.

```
dtm <- removeSparseTerms(dtm, sparse = .99)
dtm
```

```
## <<DocumentTermMatrix (documents: 2551, terms: 2061)>>
## Non-/sparse entries: 216347/5041264
## Sparsity           : 96%
## Maximal term length: 33
## Weighting          : term frequency (tf)
```

Create Training and Test Data:

```
meta <- tibble(num = dimnames(dtm)[[1]]) %>%
  left_join(spam_ham[!duplicated(spam_ham$num), ], by = "num")
```

```
set.seed(6241998)
train_index <- createDataPartition(meta$spam, p=0.80, list = FALSE, times = 1)
train <- dtm[train_index, ] %>% as.matrix() %>% as.data.frame()
test <- dtm[-train_index, ] %>% as.matrix() %>% as.data.frame()
```

Linear Support Vector Machines with Class Weights:

```
svm <- train(x = train,
                 y = as.factor(meta$spam[train_index]),
                 method = 'svmLinearWeights2',
                 trControl = trainControl(method = 'none'),
                 tuneGrid = data.frame(cost = 1, Loss = 0, weight = 1))
svm_predict <- predict(svm, newdata = test)
svm_cm <- confusionMatrix(svm_predict, as.factor(meta[-train_index, ]$spam))
svm_cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 297  69
##          1  79  64
##
##                Accuracy : 0.7092
##                  95% CI : (0.6677, 0.7484)
##     No Information Rate : 0.7387
##     P-Value [Acc > NIR] : 0.9398
##
##                   Kappa : 0.2647
##
##  Mcnemar's Test P-Value : 0.4594
```

```
##
##             Sensitivity : 0.7899
##             Specificity : 0.4812
##          Pos Pred Value : 0.8115
##          Neg Pred Value : 0.4476
##              Prevalence : 0.7387
##          Detection Rate : 0.5835
##    Detection Prevalence : 0.7191
##       Balanced Accuracy : 0.6355
##
##        'Positive' Class : 0
##
```

Naive Bayes:

```r
nb <- train(x = train,
                y = as.factor(meta$spam[train_index]),
                method = 'naive_bayes',
                trControl = trainControl(method = 'none'),
                tuneGrid = data.frame(laplace = 0, usekernel = FALSE, adjust = FALSE))
nb_predict <- predict(nb, newdata = test)
nb_cm <- confusionMatrix(nb_predict, as.factor(meta[-train_index, ]$spam))
nb_cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 313  79
##          1  63  54
##
##                Accuracy : 0.721
##                  95% CI : (0.6799, 0.7596)
##     No Information Rate : 0.7387
##     P-Value [Acc > NIR] : 0.8313
##
##                   Kappa : 0.2481
##
##  Mcnemar's Test P-Value : 0.2081
##
##             Sensitivity : 0.8324
##             Specificity : 0.4060
##          Pos Pred Value : 0.7985
##          Neg Pred Value : 0.4615
##              Prevalence : 0.7387
##          Detection Rate : 0.6149
##    Detection Prevalence : 0.7701
##       Balanced Accuracy : 0.6192
##
##        'Positive' Class : 0
##
```

Random Forest:

```
rf <- train(x = train,
                y = as.factor(meta$spam[train_index]),
                method = 'ranger',
                trControl = trainControl(method = 'none'),
                tuneGrid = data.frame(mtry = floor(sqrt(dim(train)[2])), splitrule = 'gini', min.node.s
rf_predict <- predict(rf, newdata = test)
rf_cm <- confusionMatrix(rf_predict, as.factor(meta[-train_index, ]$spam))
rf_cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 331  90
##          1  45  43
##
##                Accuracy : 0.7348
##                  95% CI : (0.6941, 0.7726)
##     No Information Rate : 0.7387
##     P-Value [Acc > NIR] : 0.6024499
##
##                   Kappa : 0.2286
##
##  Mcnemar's Test P-Value : 0.0001525
##
##             Sensitivity : 0.8803
##             Specificity : 0.3233
##          Pos Pred Value : 0.7862
##          Neg Pred Value : 0.4886
##              Prevalence : 0.7387
##          Detection Rate : 0.6503
##    Detection Prevalence : 0.8271
##       Balanced Accuracy : 0.6018
##
##        'Positive' Class : 0
##
```

| | Precision | Accuracy | Recall |
|---|---|---|---|
| SVM | 0.8114 | 0.7092 | 0.7899 |
| NB | 0.7984 | 0.721 | 0.832 |
| RF | 0.7862 | 0.7348 | 0.8803 |

Figure 1: an image caption Source: Predictions Stats.

Conclusion:

Random Forest seems to perform best out of all three models but it is very close and not one does clearly better than the rest.

I hope that with more tuning we could create a better model for predicting spam.