DT 607—Fall 2019—Project 4—Spam/Ham Classification

Team ADM - Avraham Adler, Donny Lofland, Michael Yampol

11/17/2019

Contents

Assignment

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 data-set, then predict the class of new documents (either withheld from the training data-set or from another source such as your own spam folder). One example corpus: https://spamassassin.apache.org/old/publiccorpus/

Solution

Overview

Executive Summary

The `tm` package will be used to create a corpus of data which will serve as the source of features and observations for the analysis. This will then be converted into a document-term matrix. Finally, The `caret` package will be used for the model fitting, validation, and testing.

The process of building a ham/spam filter is an oft-used pedagogical tool when teaching predictive modeling. Therefore, there is a multitude of information available on-line and in texts, of which we availed ourselves.

It should be noted that one of the more common packages in recent use for text mining, the `RTextTools` package was recently removed from CRAN, and personal communication by one of us with the author (who is now building the news feed at LinkedIn) confirmed that the package is abandonware.

Lastly, we understand that the object of this exercise is not to build an excellent predictor but to demonstrate the necessary knowledge required to build classification algorithms.

Document-Term Matrix

A document-term matrix (DTM) is the model matrix used in natural language processing (NLP). Its rows represent the documents in the corpus and its columns represent the selected terms or tokens which are treated as features. The values in each cell depends on the weighting schema selected. The simplest is *term-frequency* (tf). This is just the number of times the word is found in that document. A more sophisticated weighting scheme is *term frequency–inverse document frequency* (tf-idf). This measure increases with the frequency of the term, but offsets it by the number of documents in which it appears. This will lower the predictive power of words that naturally appear very often in all kinds of documents, and so do not shed much light on the type of document. This problem is also addressed by removing words so common as to have no predictive power at all like "and" or "the". These are often called *stop words*.

Code and Process

Style

In the following document, all user-created variables will be in `snake_case` and all user-created functions will be in `CamelCase`. Unfortunately, the `tm` packages uses `camelCase` for its functions. wE aPoLoGIze fOr anY IncoNVenIence.

Load Libraries and Set Seed

```
# allows us to repeat analysis with same outcomes
set.seed(12)

# Enable parallel processing to speed up code
library(doParallel)    # library to enable parallel processing to leverage multiple CPU's & Cores
num_cores <- detectCores() - 1

# Note that PCs , Mac and Linux need different calls to kick off multiprocessors
if(Sys.info()['sysname'] == 'Windows') {
  cl <- makePSOCKcluster(num_cores, type="FORK")
} else {
  cl <- makeCluster(num_cores, type="FORK")
}

registerDoParallel(cl)

library(tm)            # tool to facilitate building corpus of data
library(SnowballC)     # tools to find word stems
library(caret)         # tools to run machine learning
library(wordcloud)     # tool to help build vidual wordclouds
library(tidyverse)
```

List files

The files were downloaded from the link above, and the `spam_2` and `easy_ham` sets were selected for analysis. These were unzipped so that each email is its own file in the directory.

```
# Get a list of all the spam file names (each file is a single email message)
s_files <- list.files("./Data/spam_2", full.names = TRUE)
s_len <- length(s_files)

# Get a list of all the ham files names (each file is a single email message)
h_files <- list.files("./Data/easy_ham", full.names = TRUE)
h_len <- length(h_files)
```

We loaded `{r} s_len` spam email messages and `{r} h_len` ham (non-spam) email messages. The first thing to note is that we have an unbalanced data set with more good email messages (ham) than spam. This may affect our choice of models and/or force us to take extra steps to accomodate the difference in set sizes.

Building the Corpus

Email Headers

We will be focusing on email content, and not the meta information or doing reverse DNS lookups. Therefore, it makes sense to remove the email headers. According to the most recent RFC about email, RFC 5322, Section 2.2, the header should not contain any purely blank lines. Therefore, it is a very reasonable approach to look for the first blank line and only start ingesting the email from the next line. That is what is searched for by the regex pattern `"^$"` in the function below.

In the headers, some information that could be used to enhance a model might include: the Subject line, sender's email address domain name (e.g. @gmail.com, @companyname.com, etc), whether the sender's email domain matches the sender's SMTP server domain name, the hour (UTC) when the email was sent, the origin country (based on SMTP server name or IP address lookup), and potentially information about the originating domain name (e.g. when was he domain registered). If this was a critical project, we could also download RBL (realtime blakc lists) and use that information to provide additional pattern matching.

Raw Corpus

The `readLines` function reads each line as a separate vector. To turn this into a single character vector, the `paste` function is used with the appropriate `sep` and `collapse` values. The class of the document is passed as a parameter to the `BuildCorpus` function.

```
#' Build a corpus from a list of file names
#'
#' @param files List of documents to load.
#' @param class The class to be applied to the loaded documents
#' @return A charater vector
BuildCorpus <- function(files, class) {

  # loop thru files and process each one as we go
  for (i in seq_along(files)) {
    raw_text <- readLines(files[i])
    em_length <- length(raw_text)

    # Lets extract the Subject line (if present) and clean it
    subject_line <- str_extract(raw_text, "^Subject: (.*)$")
    subject_line <- subject_line[!is.na(subject_line)]
# Note that PCs do not need this
    if(Sys.info()['sysname'] != 'Windows') {
      subject_line <- iconv(subject_line, to="UTF-8")
    }

    # let's scrub / clean up the subject line text
    subject_line <- gsub("[^0-9A-Za-z///' ]","" , subject_line, ignore.case = TRUE, useBytes = TRUE)
    subject_line <- tolower(subject_line)
    subject_line <- str_replace_all(subject_line, "(\\[)|(\\])|(re )|(subject )", "")

    # Lets extract the email body content
    body_start <- min(grep("^$", raw_text, fixed = FALSE, useBytes = TRUE)) + 1L
    em_body <- paste(raw_text[body_start:em_length], sep="", collapse=" ")
# Note that PCs do not need this
    if(Sys.info()['sysname'] != 'Windows') {
      em_body <- iconv(em_body, to="UTF-8")
```

```
    }
    # make the text lower case
    em_body <- tolower(em_body)

    # remove HTML tags
    em_body <- str_replace_all(em_body, "(<[^>]*>)", "")
    em_body <- str_replace_all(em_body, "(&.*;)", "")

    # remove any URL's
    em_body <- str_replace_all(em_body, "http(s)?:(.*) ", " ")

    # remove non alpha (leave lower case and apostrophe for contractions)
    em_body <- str_replace_all(em_body, "[^a-z///' ]", "")
    em_body <- str_replace_all(em_body, "''|' ", "")

    # Since the subject line might have important info, lets concatenate it to the top of the email bod
    em_body <- paste(c(subject_line, em_body), sep="", collapse=" ")

    if (i == 1L) {
      ret_Corpus <- VCorpus(VectorSource(em_body))
    } else {
      tmp_Corpus <- VCorpus(VectorSource(em_body))
      ret_Corpus <- c(ret_Corpus, tmp_Corpus)
    }
  }

  meta(ret_Corpus, tag = "class", type = "indexed") <- class

  return(ret_Corpus)
}

h_corp_raw <- BuildCorpus(h_files, "ham")
s_corp_raw <- BuildCorpus(s_files, "spam")
```

Cleaning the Corpus

We used many of the default cleaning tools in the `tm` package to perform standard adjustments like lower-casing, removing numbers, etc. We made two non-native adjustments. First we stripped out anything that looked like a URL. This needed to be done prior to removing punctuation, of course. We also added a few words to the removal list which we think have little predictive power due to their overuse. We considered removing all punctuation, but decided to leave both intra-word contractions and internal punctuation.

Lastly, we used the `SnowballC` package to stem the document. This process tries to identify common roots shared by similar words and then treat them as one. For example:

```
wordStem(c('run', 'running', 'ran', 'runt'), language = 'porter')
```

```
## [1] "run"  "run"  "ran"  "runt"
```

The complete cleaning rules are in the `CleanCorpus` function.

```
# https://stackoverflow.com/questions/47410866/r-inspect-document-term-matrix-results-in-error-repeated
#' Scrub the text in a corpus
#' @param corpus A text corpus prepared by tm
#' @return A sanitized corpus
CleanCorpus <- function(corpus){
  overused_words <- c("ok", 'okay', 'day', "might", "bye", "hello", "hi",
                      "dear", "thank", "you", "please", "sorry")

  # lower case everything
  corpus <- tm_map(corpus, content_transformer(tolower))

  # remove any HTML markup
  removeHTMLTags <- function(x) {gsub("(<[^>]*>)", "", x)}
  corpus <- tm_map(corpus, content_transformer(removeHTMLTags))

  # remove any URL's
  StripURL <- function(x) {gsub("(http[^ ]*)|(www\\.[^ ]*)", "", x)}
  corpus <- tm_map(corpus, content_transformer(StripURL))

  # remove anything not a simple letter
  KeepAlpha <- function(x) {gsub("[^a-z///-///' ]", "", x, ignore.case = TRUE, useBytes = TRUE)}
  corpus <- tm_map(corpus, content_transformer(KeepAlpha))

  # remove any numbers
  corpus <- tm_map(corpus, removeNumbers)

  # remove punctuation
  corpus <- tm_map(corpus, removePunctuation,
                   preserve_intra_word_contractions = TRUE,
                   preserve_intra_word_dashes = TRUE)

  # remove any stop words
  corpus <- tm_map(corpus, removeWords, stopwords("english"))
  corpus <- tm_map(corpus, removeWords, overused_words)

  # remove extra white space
```

```r
  corpus <- tm_map(corpus, stripWhitespace)

  # use the SnowballC stem algorithm to find the root stem of similar words
  corpus <- tm_map(corpus, stemDocument)

  return(corpus)
}
```

Removing Very Sparse Terms

Even with a cleaned corpus, the overwhelming majority of the terms are rare. There are two ways to address sparsity of terms in the `tm` package. The first is to generate a list of words that appear at least k times in the corpus. This is done using the `findFreqTerms` command. Then the document-term matrix (DTM) can be built using only those words.

The second way is to build the DTM with all words, and then remove the words that don't appear in at least $p\%$ of documents. This is done using the `removeSparseTerms` function in `tm`. Both methods make manual inspection of more than one line of the matrix impossible. The matrix is stored sparsely as a triplet, and once terms are removed, it becomes impossible for R to print properly.

The `removeSparseTerms` is intuitively more appealing as it measures frequency by document, and not across documents. However, applying that to three separate corpuses would result in the validation and testing sets not having the same words as the training set. Therefore, the build-up method will be used, but used by finding the remaining terms after calling remove.

However, before we do that, we need to discuss…

Training, Validation, and Testing

Hastie & Tibshirani, in their seminal work ESL, suggest breaking ones data into three parts: 50% training, 25% validation, and 25% testing. Confusingly, some literature uses "test" for the validation set and "holdout" for the test set. Regardless, the idea is that you train your model on 50% of the data, and use 25% of the data (the validation set) to refine any hyper-parameters of the model. You do this for each model, and then once all the models are tuned as best possible, they are compared with each other by their performance on the heretofore unused testing/holdout set. The `SplitSample` function was used to split the data at the start.

```r
# https://stackoverflow.com/questions/47410866/r-inspect-document-term-matrix-results-in-error-repeated
#' Split a sample into Training, Validation and Test groups.
#' Return a vector with the label for each sample using the provided probabilities.
#' Note: training, validation and test should be non-negative and, not all zero.
#' @param n The total number of samples in the set
#' @param n Desired training set size (percent)
#' @param n Desired validation set size (percent)
#' @param n Desired test set size (percent)
#' @return A sanitized corpus
SplitSample <- function(n, training=0.5, validation=0.25, test=0.25) {
  if((training >= 0 && validation >= 0 && test >= 0) &&
     ((training + validation + test) > 0) &&
     ((training + validation + test) <= 1.0 )) {
    n_split <- sample(x = c("train", "validate", "test"), size = n,
                      replace = TRUE, prob = c(0.5, 0.25, 0.25))
  } else {
    n_split <- FALSE
  }

  return(n_split)
}

# build vectors that identify which group each sample will be placed (training, validation or test)
h_split <- SplitSample(h_len)
s_split <- SplitSample(s_len)
```

Note that with machine learning, another popular approach is to setup **K-fold Cross Validation**. With this approach, we create a Training/Testing split as shown above, train a model, then repeat the process with a different random Training/Testing splits. By iterating (typically 5-10 times), we ensure that every observation has a chance of being included during Training or Testing and can appear in any split group. We then average the performance metrics and use that to evaluate the model. This helps reduce bias that might have been introduced by random chance with just a single Training/Testing split.

If there are limited number of samples to work with, thus limiting the information available during the training phase, it is common to compromise and use a 70%/30% or 80%/20% Training to Testing split and skip the third Validation set. If there are limited observations, *Bootstrapping* is one method for generating additional data and works well if the known samples provide sufficient reprentation of the expected distribution of possible values or datapoints.

When we have the possibility of multiple rows from the same source, there is the possibility of leakage between the training and test/validation sets such that the model performs better on the validation and/or test sets than expected. We are not going to consider this now, but a more rigorous model would tag each row with the sender's email address and/or IP address and use `groupKFold()` or some other similar technique to ensures all rows from a given sender are kept together in the same data set (trainng, validation or test). See https://topepo.github.io/caret/data-splitting.html for more information. Note that this approach can lead to complexity . . . for further discussion, see https://towardsdatascience.com/the-story-of-a-bad-train-test-split-3343fcc33d2c.

Building the Term List

As both training and validation are part of the model construction, we feel that the term list can be built from the combination of the two. The terms in the testing/holdout set will not be seen prior to testing. We will restrict the word list to words that appear in at least 100 of the combined 2922 documents. In a real world scenario, email messages may contain new terms not seen suring the training steps. By excluding the final validation terms, we better simulate a realworld implementation where new words are appearing that we didn't have available during model training

```
# pull all terms from the training sets (both hame and spam)
raw_train <- c(h_corp_raw[h_split == "train"],
               s_corp_raw[s_split == "train"])

# pull all terms from the validation sets (both hame and spam)
raw_val <- c(h_corp_raw[h_split == "validate"],
             s_corp_raw[s_split == "validate"])

# pull all terms from the test sets (both hame and spam)
raw_test <- c(h_corp_raw[h_split == "test"],
              s_corp_raw[s_split == "test"])

# combine both training and test terms into a master list
raw_term_corp <- c(raw_train, raw_val)
clean_term_corp <- CleanCorpus(raw_term_corp)

dtm_terms <- DocumentTermMatrix(clean_term_corp, control = list(bounds = list(global = c(100L, Inf))))

freq_terms <- Terms(dtm_terms)
```

Here are the top 20 stemmed terms out of the 308 terms we will use in the dictionary:

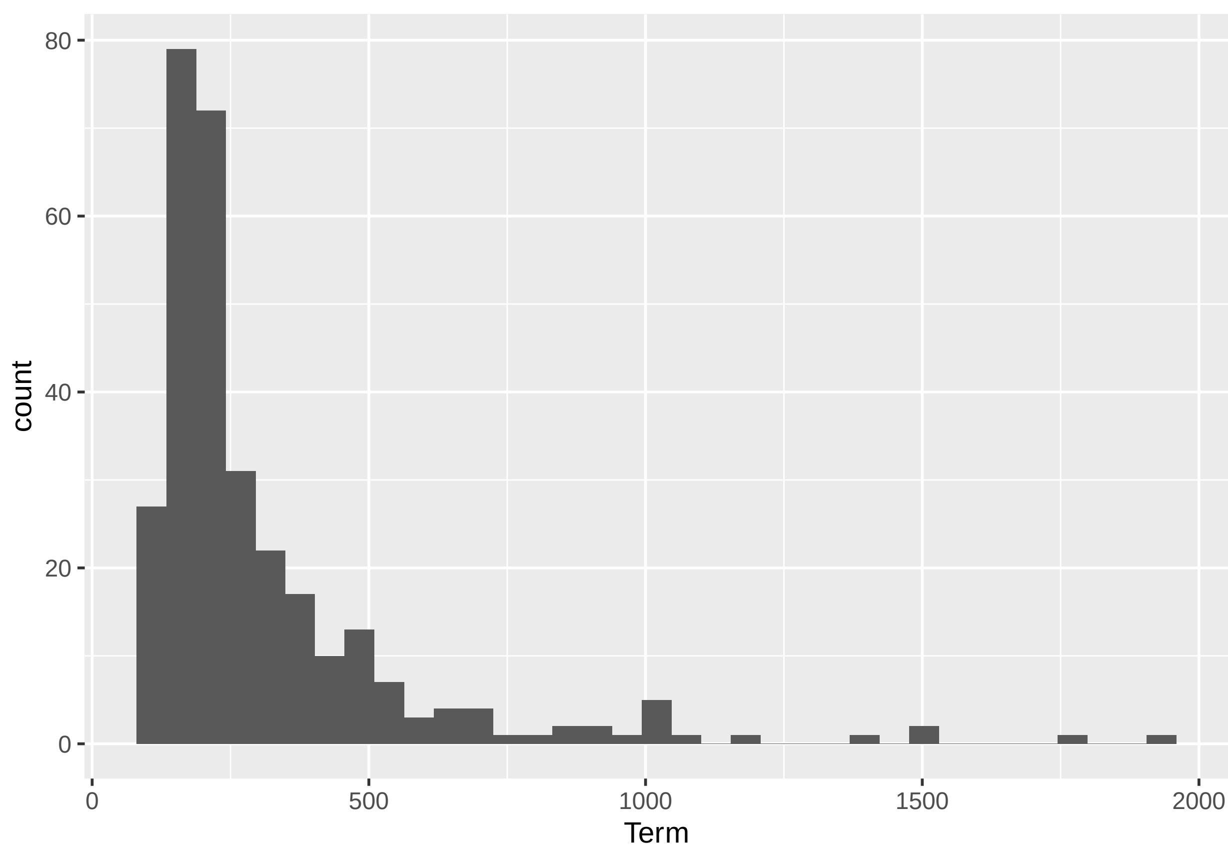
```
ft <- colSums(as.matrix(dtm_terms))
ft_df <- data.frame(term = names(ft), count = as.integer(ft))
knitr::kable(head(ft_df[order(ft, decreasing = TRUE), ], n = 20L),
             row.names = FALSE)
```

term	count
email	1914
will	1757
use	1499
can	1490
get	1416
one	1196
just	1075
mail	1039
free	1023
time	1012
work	1003
list	1002
messag	967
like	908
make	899

term	count
now	858
peopl	834
new	815
receiv	755
order	677

Here is a histogram of word frequency using the Freedman-Diaconis rule for binwidth.

```
bw_fd <- 2 * IQR(ft_df$count) / (dim(ft_df)[[1]]) ^ (1/3)
ggplot(ft_df, aes(x = count)) + geom_histogram(binwidth = bw_fd) + xlab("Term")
```



Finally, a wordcloud of the stemmed terms appearing at least 250 times:

```
wordcloud(ft_df$term,ft_df$count, scale = c(3, 0.6), min.freq = 250L,
          colors = brewer.pal(5, "Dark2"), random.color = TRUE,
          random.order = TRUE, rot.per = 0, fixed.asp = FALSE)
```


year get state address
sinc offer
everi program receiv save
take
softwar url system just report write
subject
compani mail product messag file
wrote
follow sure includ name like
start servic can rate best contenttyp instal
say without also world
free now order chang
test
way busi phone
look send may thing
differ
know
help spam market week per credit click good
line internet
home think even run come code packag set someth
within
sent give
life million
money interest
work will make well
much site
web provid
seem base first right person
peopl back tri find new time read said two price
today mani
need realli
list inform call still want check
email one problem see
remov anyon build sponsor server
month use
number

Building the Training Set

```
# sample is to randomize the observations
clean_train <- sample(CleanCorpus(raw_train))
clean_train_type <- unlist(meta(clean_train, tag = "class"))
attributes(clean_train_type) <- NULL
dtm_train <- DocumentTermMatrix(clean_train,
                                control = list(dictionary = freq_terms))
dtm_train
```

```
## <<DocumentTermMatrix (documents: 1943, terms: 308)>>
## Non-/sparse entries: 42190/556254
## Sparsity           : 93%
## Maximal term length: 20
## Weighting          : term frequency (tf)
```

Compare the above with the sparsity of the cleaned training corpus without the limiting dictionary:

```
dtm_train_S <- DocumentTermMatrix(clean_train)
dtm_train_S
```

```
## <<DocumentTermMatrix (documents: 1943, terms: 19211)>>
## Non-/sparse entries: 114275/37212698
## Sparsity           : 100%
## Maximal term length: 441
## Weighting          : term frequency (tf)
```

Building the Validation Set

```
clean_val <- sample(CleanCorpus(raw_val))
clean_val_type <- unlist(meta(clean_val, tag = "class"))
attributes(clean_val_type) <- NULL
dtm_val <- DocumentTermMatrix(clean_val,
                              control = list(dictionary = freq_terms))
```

Building the Testing Set

```
clean_test <- sample(CleanCorpus(raw_test))
clean_test_type <- unlist(meta(clean_test, tag = "class"))
attributes(clean_test_type) <- NULL
dtm_test <- DocumentTermMatrix(clean_test,
                              control = list(dictionary = freq_terms))
```

Last step

The `caret` package requires its input to be a numeric matrix. As the DTM is a special form of sparse matrix, we need to convert it to something `caret` understands. The response vector must be a factor for classification, which is why all three `clean_x_type` vectors were created as factors.

```r
train_m <- as.matrix(dtm_train)
clean_train_type <- factor(clean_train_type, levels = c("spam", "ham"))
val_m <- as.matrix(dtm_val)
clean_val_type <- factor(clean_val_type, levels = c("spam", "ham"))
test_m <- as.matrix(dtm_test)
clean_test_type <- factor(clean_test_type, levels = c("spam", "ham"))
```

Train Models

Overview

Now we can train the models. The process will generally follow the following path:

1. Select a model family (logistic regression, random forest, etc.)
2. Use the `caret` package on the training set to pick "best" model given the supplied control, pre-processing, or other [hyper-]parameters. This may include some level of validation
3. Switch the hyper-parameters, train again, and compare using validation set
4. Select "best" model from family
5. Repeat with other families
6. Compare performance of final selections using testing/holdout set
7. Take a well-deserved vacation

As the `caret` package serves as an umbrella for over 230 model types living in different packages, we may select a less-sophisticated version of a family if it reduces code complexity and migraine propensity. Forgive us as well if we don't explain every family and every selection. Below we create the model matrices which will be passed to `caret`.

Experimentation was done with many of the tuning parameters. However, most increases in accuracy came at an inordinate expense of time. Therefore, for the purposes of this exercise, many of the more advantageous options will be limited. For example, cross-validation will be limited to single-pass ten-fold. In production, one should be more vigorous, of course.

Optimization Metric

Usually, AUC, a function of ROC, is used for classification problems. However, for imbalanced data sets it is suggested to use one of precision, recall, or F1 instead. See here, here, or here for examples.

In our case, the data set is imbalanced, and the cost of a false positive (classifying ham as spam) is greater than a false negative. Originally, we selected precision as the metric, as hitting the "junk" button for something in your inbox is less annoying than having your boss's email sit in your junk folder.

However, as we trained models, we found some fascinating results. In one of the random forest models, the algorithm found a better model with one less false positive, at the expense of 61 more false negatives. Therefore, we decided to redo the tests using the balanced F1 as the optimization metric.

Logistic Regression

This is the classic good-old logistic regression in R. There are no hyper/tuning parameters, so the only comparison can be between the method of cross-validation.

```r
# 10-fold CV
tr_ctrl <- trainControl(method = "cv", number = 10L, classProbs = TRUE,
                        summaryFunction = prSummary)
LogR1 <- train(x = train_m, y = clean_train_type, method = "glm",
              family = "binomial", trControl = tr_ctrl, metric = "F", model=TRUE)
LogR1
```

```
## Generalized Linear Model
##
## 1943 samples
##  308 predictor
##    2 classes: 'spam', 'ham'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1749, 1748, 1749, 1749, 1749, 1748, ...
## Resampling results:
##
##   AUC         Precision  Recall     F
##   0.06516546  0.8580168  0.9107246  0.883002
```

```r
LogR1v <- predict(LogR1, val_m)
confusionMatrix(LogR1v, clean_val_type, mode = "prec_recall", positive = "spam")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction spam ham
##       spam  320  42
##       ham    32 553
##
##                Accuracy : 0.9219
##                  95% CI : (0.9029, 0.9381)
##     No Information Rate : 0.6283
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8337
##
##  Mcnemar's Test P-Value : 0.2955
##
##               Precision : 0.8840
##                  Recall : 0.9091
##                      F1 : 0.8964
##              Prevalence : 0.3717
##          Detection Rate : 0.3379
##    Detection Prevalence : 0.3823
##       Balanced Accuracy : 0.9193
##
```

```
##        'Positive' Class : spam
##
```

```
# Monte-Carlo Cross validation using 75/25 and 5 iterations
tr_ctrl <- trainControl(method = "LGOCV", number = 10L, p = 0.75,
                        classProbs = TRUE, summaryFunction = prSummary)
LogR2 <- train(x = train_m, y = clean_train_type, method = "glm",
              family = "binomial", trControl = tr_ctrl, metric = "F", model=TRUE)
LogR2
```

```
## Generalized Linear Model
##
## 1943 samples
##  308 predictor
##    2 classes: 'spam', 'ham'
##
## No pre-processing
## Resampling: Repeated Train/Test Splits Estimated (10 reps, 75%)
## Summary of sample sizes: 1458, 1458, 1458, 1458, 1458, 1458, ...
## Resampling results:
##
##   AUC         Precision  Recall     F
##   0.07648102  0.8440346  0.8890173  0.8647148
```

```
LogR2v <- predict(LogR2, val_m)
confusionMatrix(LogR2v, clean_val_type, mode = "prec_recall", positive = "spam")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction spam ham
##       spam  320  42
##       ham    32 553
##
##                Accuracy : 0.9219
##                  95% CI : (0.9029, 0.9381)
##     No Information Rate : 0.6283
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8337
##
##  Mcnemar's Test P-Value : 0.2955
##
##               Precision : 0.8840
##                  Recall : 0.9091
##                      F1 : 0.8964
##              Prevalence : 0.3717
##          Detection Rate : 0.3379
##    Detection Prevalence : 0.3823
##       Balanced Accuracy : 0.9193
##
##        'Positive' Class : spam
##
```

Both versions performed the same on the validation set. As the first has a slightly better F-score, we will select that one.

Feature importance

Which terms had the most influence on ham/spam classification using Logistic Regression?

```r
# estimate variable importance
importance <- varImp(LogR2)
# summarize importance
print(importance)
```

```
## glm variable importance
##
##   only 20 most important variables shown (out of 308)
##
##                      Overall
## post                  100.00
## url                    97.92
## click                  97.70
## wrote                  90.27
## want                   74.61
## futur                  65.49
## server                 65.43
## seem                   62.07
## credit                 59.33
## visit                  58.91
## contenttransferencod   58.85
## write                  58.49
## two                    57.40
## use                    56.40
## error                  54.31
## dollar                 48.78
## type                   47.91
## peopl                  47.76
## test                   47.69
## linux                  47.47
```

Random Forest

The `ranger` package is used as the random forest engine due to its being optimized for higher dimensions.

```
tr_ctrl <- trainControl(method = "cv", number = 10L, classProbs = TRUE,
                        summaryFunction = prSummary)
RF1 <- train(x = train_m, y = clean_train_type, method = 'ranger', importance = 'impurity',
             trControl = tr_ctrl, metric = "F", tuneLength = 5L)
RF1
```

```
## Random Forest
##
## 1943 samples
##  308 predictor
##    2 classes: 'spam', 'ham'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1750, 1748, 1749, 1748, 1748, 1748, ...
## Resampling results across tuning parameters:
##
##   mtry  splitrule   AUC        Precision  Recall     F
##     2   gini        0.9598693  0.9802080  0.8417598  0.9050521
##     2   extratrees  0.9589222  0.9828103  0.8072878  0.8853248
##    78   gini        0.9076124  0.9150241  0.9194410  0.9162988
##    78   extratrees  0.9395113  0.9243902  0.9338509  0.9281870
##   155   gini        0.8245228  0.9014651  0.9223188  0.9109659
##   155   extratrees  0.8950962  0.9073504  0.9324224  0.9189829
##   231   gini        0.7535807  0.8922116  0.9180538  0.9042518
##   231   extratrees  0.8760631  0.8886891  0.9338923  0.9101560
##   308   gini        0.6631668  0.8874890  0.9223188  0.9040437
##   308   extratrees  0.8388397  0.8912690  0.9338716  0.9114358
##
## Tuning parameter 'min.node.size' was held constant at a value of 1
## F was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 78, splitrule =
##  extratrees and min.node.size = 1.
```

```
RF1v <- predict(RF1, newdata=val_m)
confusionMatrix(RF1v, clean_val_type, mode = "prec_recall", positive = "spam")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction spam ham
##       spam  325  18
##       ham    27 577
##
##                Accuracy : 0.9525
##                  95% CI : (0.9369, 0.9651)
##     No Information Rate : 0.6283
##     P-Value [Acc > NIR] : <2e-16
##
```

```
##                   Kappa : 0.8977
##
##  Mcnemar's Test P-Value : 0.233
##
##               Precision : 0.9475
##                  Recall : 0.9233
##                      F1 : 0.9353
##              Prevalence : 0.3717
##          Detection Rate : 0.3432
##    Detection Prevalence : 0.3622
##       Balanced Accuracy : 0.9465
##
##        'Positive' Class : spam
##
```

Let's do a bit wider search among tuning parameters.

```
rf_grid <- expand.grid(mtry = seq(8, 48, 4),
                       splitrule = c('gini', 'extratrees'),
                       min.node.size = c(1L, 10L))
RF2 <- train(x = train_m, y = clean_train_type, method = 'ranger', importance = 'impurity',
             trControl = tr_ctrl, metric = "F", tuneGrid = rf_grid)
RF2
```

```
## Random Forest
##
## 1943 samples
##  308 predictor
##    2 classes: 'spam', 'ham'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1748, 1748, 1749, 1748, 1749, 1749, ...
## Resampling results across tuning parameters:
##
##   mtry  splitrule   min.node.size  AUC        Precision  Recall
##    8    gini         1             0.9705911  0.9633420  0.9152381
##    8    gini        10             0.9683954  0.9659951  0.9195445
##    8    extratrees   1             0.9711358  0.9719806  0.9238302
##    8    extratrees  10             0.9699666  0.9733659  0.9195445
##   12    gini         1             0.9709628  0.9494133  0.9267288
##   12    gini        10             0.9681634  0.9520939  0.9224224
##   12    extratrees   1             0.9698586  0.9665863  0.9353002
##   12    extratrees  10             0.9689399  0.9621027  0.9324224
##   16    gini         1             0.9692627  0.9464625  0.9252795
##   16    gini        10             0.9688930  0.9452474  0.9209731
##   16    extratrees   1             0.9697263  0.9580291  0.9353209
##   16    extratrees  10             0.9685598  0.9634972  0.9309731
##   20    gini         1             0.9689225  0.9411957  0.9281573
##   20    gini        10             0.9668867  0.9393335  0.9209524
##   20    extratrees   1             0.9678762  0.9540518  0.9381781
##   20    extratrees  10             0.9708979  0.9607867  0.9352795
##   24    gini         1             0.9687893  0.9386130  0.9252795
##   24    gini        10             0.9694745  0.9370235  0.9223810
```

```
##   24    extratrees   1             0.9695748  0.9551435  0.9367288
##   24    extratrees  10             0.9687834  0.9580662  0.9338302
##   28    gini         1             0.9693475  0.9363096  0.9281159
##   28    gini        10             0.9663106  0.9313166  0.9238095
##   28    extratrees   1             0.9699114  0.9555167  0.9410352
##   28    extratrees  10             0.9688853  0.9534194  0.9309731
##   32    gini         1             0.9617796  0.9289073  0.9295859
##   32    gini        10             0.9670533  0.9298420  0.9209317
##   32    extratrees   1             0.9678589  0.9510896  0.9381781
##   32    extratrees  10             0.9681864  0.9532061  0.9295238
##   36    gini         1             0.9623117  0.9298491  0.9266874
##   36    gini        10             0.9673603  0.9287059  0.9223602
##   36    extratrees   1             0.9661140  0.9418472  0.9381781
##   36    extratrees  10             0.9689722  0.9511305  0.9338509
##   40    gini         1             0.9560490  0.9235379  0.9252588
##   40    gini        10             0.9655757  0.9248219  0.9238095
##   40    extratrees   1             0.9621528  0.9438617  0.9324431
##   40    extratrees  10             0.9676265  0.9451817  0.9309731
##   44    gini         1             0.9486435  0.9230755  0.9238302
##   44    gini        10             0.9656740  0.9273154  0.9238095
##   44    extratrees   1             0.9606299  0.9425747  0.9338923
##   44    extratrees  10             0.9691245  0.9397354  0.9266667
##   48    gini         1             0.9416125  0.9217636  0.9238095
##   48    gini        10             0.9619096  0.9205998  0.9252381
##   48    extratrees   1             0.9613769  0.9359074  0.9338302
##   48    extratrees  10             0.9689179  0.9427401  0.9295445
##   F
##   0.9377567
##   0.9416071
##   0.9467500
##   0.9450408
##   0.9374036
##   0.9363567
##   0.9501963
##   0.9465758
##   0.9352125
##   0.9321951
##   0.9459583
##   0.9463515
##   0.9340683
##   0.9294181
##   0.9455800
##   0.9474372
##   0.9311222
##   0.9289741
##   0.9454387
##   0.9453289
##   0.9319654
##   0.9270296
##   0.9477620
##   0.9416684
##   0.9286565
##   0.9247470
##   0.9442013
```

```
##   0.9408922
##   0.9277462
##   0.9249851
##   0.9395133
##   0.9419858
##   0.9237209
##   0.9236287
##   0.9377670
##   0.9377047
##   0.9228945
##   0.9250193
##   0.9378640
##   0.9327154
##   0.9222807
##   0.9223418
##   0.9344874
##   0.9355756
##
## F was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 12, splitrule =
##  extratrees and min.node.size = 1.
```

```r
RF2v <- predict(RF2, val_m)
confusionMatrix(RF2v, clean_val_type, mode = "prec_recall", positive = "spam")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction spam ham
##       spam  325  10
##       ham    27 585
##
##                Accuracy : 0.9609
##                  95% CI : (0.9465, 0.9723)
##     No Information Rate : 0.6283
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9155
##
##  Mcnemar's Test P-Value : 0.008529
##
##               Precision : 0.9701
##                  Recall : 0.9233
##                      F1 : 0.9461
##              Prevalence : 0.3717
##          Detection Rate : 0.3432
##    Detection Prevalence : 0.3537
##       Balanced Accuracy : 0.9532
##
##        'Positive' Class : spam
##
```

Interestingly, the first model performed better on the validation set despite performing more poorly on the training set. Possibly an example of overfitting.

Feature importance

Which terms had the most influence on ham/spam classification using Random Forest?

```r
# estimate variable importance
importance <- varImp(RF2)
# summarize importance
print(importance)
```

```
## ranger variable importance
##
##   only 20 most important variables shown (out of 308)
##
##                     Overall
## click               100.000
## url                  61.274
## wrote                38.717
## remov                37.468
## visit                23.094
## free                 21.810
## receiv               20.975
## contenttransferencod  19.164
## credit               17.997
## guarante             16.511
## email                16.419
## inform               12.955
## unsubscrib           12.006
## contact              11.496
## use                  11.304
## repli                11.269
## offer                10.977
## contenttyp           10.912
## life                  9.995
## onlin                 9.792
```

Naive Bayes

```r
tr_ctrl <- trainControl(method = "cv", number = 10L, classProbs = TRUE,
                        summaryFunction = prSummary)
NB1 <- train(x = train_m, y = clean_train_type, method = "nb",
             trControl = tr_ctrl, metric = "F")
NB1
```

```
## Naive Bayes
##
## 1943 samples
##  308 predictor
##    2 classes: 'spam', 'ham'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1750, 1749, 1748, 1750, 1748, 1749, ...
## Resampling results across tuning parameters:
##
##   usekernel  AUC        Precision  Recall      F
##   FALSE            NaN  NaN               NaN         NaN
##    TRUE      0.8645713    1        0.03610766  0.06863714
##
## Tuning parameter 'fL' was held constant at a value of 0
## Tuning
##  parameter 'adjust' was held constant at a value of 1
## F was used to select the optimal model using the largest value.
## The final values used for the model were fL = 0, usekernel = TRUE
##  and adjust = 1.
```

```r
NB1v <- predict(NB1, val_m)
confusionMatrix(NB1v, clean_val_type, mode = "prec_recall", positive = "spam")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction spam ham
##       spam   13   2
##       ham   339 593
##
##                Accuracy : 0.6399
##                  95% CI : (0.6084, 0.6705)
##     No Information Rate : 0.6283
##     P-Value [Acc > NIR] : 0.2405
##
##                   Kappa : 0.0417
##
##  Mcnemar's Test P-Value : <2e-16
##
##               Precision : 0.86667
##                  Recall : 0.03693
##                      F1 : 0.07084
```

```
##               Prevalence : 0.37170
##           Detection Rate : 0.01373
##     Detection Prevalence : 0.01584
##        Balanced Accuracy : 0.51679
##
##         'Positive' Class : spam
##
```

This is an *awfully* performing model. Naive Bayes is known to be very sensitive to class imbalances. Let's implement up-sampling and a wider search.

```
tr_ctrl <- trainControl(method = "cv", number = 10L, classProbs = TRUE,
                        summaryFunction = prSummary, sampling = 'up')
nb_grid <- expand.grid(usekernel = TRUE,
                       fL = seq(0.25, 0.75, 0.05),
                       adjust = 1)
NB2 <- train(x = train_m, y = clean_train_type, method = "nb",
             trControl = tr_ctrl, metric = "F", tuneGrid = nb_grid)
NB2
```

```
## Naive Bayes
##
## 1943 samples
##  308 predictor
##    2 classes: 'spam', 'ham'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1749, 1749, 1749, 1749, 1749, 1749, ...
## Addtional sampling using up-sampling
##
## Resampling results across tuning parameters:
##
##   fL    AUC        Precision  Recall      F
##   0.25  0.8417755  0.9088889  0.08900621  0.1597567
##   0.30  0.8345597  0.8491176  0.10627329  0.1795763
##   0.35  0.8229549  0.8546093  0.11211180  0.1845699
##   0.40  0.8362909  0.8916667  0.10494824  0.1793880
##   0.45  0.8338190  0.9133333  0.10354037  0.1826036
##   0.50  0.8302064  0.8875000  0.10068323  0.1741349
##   0.55  0.8301797  0.9196032  0.10339545  0.1750240
##   0.60  0.8222250  0.8914286  0.10194617  0.1760769
##   0.65  0.8371354  0.9168831  0.10927536  0.1859311
##   0.70  0.8364672  0.9524510  0.10486542  0.1809627
##   0.75  0.8335795  0.9051535  0.09774327  0.1680177
##
## Tuning parameter 'usekernel' was held constant at a value of TRUE
##
## Tuning parameter 'adjust' was held constant at a value of 1
## F was used to select the optimal model using the largest value.
## The final values used for the model were fL = 0.65, usekernel = TRUE
##  and adjust = 1.
```

```r
NB2v <- predict(NB2, val_m)
confusionMatrix(NB2v, clean_val_type, mode = "prec_recall", positive = "spam")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction spam ham
##       spam   36   2
##       ham   316 593
##
##                Accuracy : 0.6642
##                  95% CI : (0.6331, 0.6943)
##     No Information Rate : 0.6283
##     P-Value [Acc > NIR] : 0.01174
##
##                   Kappa : 0.1209
##
##  Mcnemar's Test P-Value : < 2e-16
##
##               Precision : 0.94737
##                  Recall : 0.10227
##                      F1 : 0.18462
##              Prevalence : 0.37170
##          Detection Rate : 0.03801
##    Detection Prevalence : 0.04013
##       Balanced Accuracy : 0.54946
##
##        'Positive' Class : spam
##
```

Results are still **miserable**. Naive Bayes also assumes **Independence** between all features - with engligh text, words/terms are likely to have correlations thus violating the core assumption of Naive Bayes. Since our current terms also some leakage of HTML tags and attributes, there are going to be correlations between terms we have selected. Naive Bayes would probably perform significantly better if we stipped all HTML terms and made a pass on reducing features by looking for correlations.

Feature importance

Which terms had the most influence on ham/spam classification using Naive Bayes?

```r
# estimate variable importance
importance <- varImp(NB2)
# summarize importance
print(importance)
```

```
## ROC curve variable importance
##
##   only 20 most important variables shown (out of 308)
##
##                      Importance
## click                    100.00
## email                     91.87
## remov                     72.56
## wrote                     72.45
## receiv                    68.95
## free                      61.85
## will                      59.82
## url                       54.24
## inform                    51.45
## busi                      43.61
## address                   42.60
## offer                     42.37
## repli                     35.31
## money                     35.17
## now                       32.42
## contenttransferencod      32.36
## can                       32.21
## month                     31.56
## send                      30.95
## contenttyp                30.15
```

Neural Network

```
tr_ctrl <- trainControl(method = "cv", number = 10L, classProbs = TRUE,
                        summaryFunction = prSummary)

NN1 <- train(x = train_m, y = clean_train_type, method = "nnet", trace = FALSE,
             trControl = tr_ctrl, metric = "F", tuneLength=5L, maxit = 250L)
NN1
```

```
## Neural Network
##
## 1943 samples
##  308 predictor
##    2 classes: 'spam', 'ham'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1749, 1748, 1750, 1749, 1749, 1749, ...
## Resampling results across tuning parameters:
##
##   size  decay  AUC        Precision  Recall     F
##   1     0e+00  0.1938775  0.9151743  0.9108489  0.9124544
##   1     1e-04  0.3399736  0.9214103  0.9195238  0.9196794
##   1     1e-03  0.4819758  0.9196001  0.9136853  0.9156806
##   1     1e-02  0.6758812  0.9178383  0.9236853  0.9198730
##   1     1e-01  0.8356265  0.9276069  0.9352588  0.9307978
##   3     0e+00  0.2302942  0.9203843  0.9424431  0.9310746
##   3     1e-04  0.3270280  0.9165794  0.9309938  0.9231026
##   3     1e-03  0.4908054  0.9227723  0.9165217  0.9191806
##   3     1e-02  0.9345542  0.9205250  0.9223188  0.9211412
##   3     1e-01  0.9622627  0.9375728  0.9280538  0.9321527
##   5     0e+00        NaN        NaN        NaN        NaN
##   5     1e-04        NaN        NaN        NaN        NaN
##   5     1e-03        NaN        NaN        NaN        NaN
##   5     1e-02        NaN        NaN        NaN        NaN
##   5     1e-01        NaN        NaN        NaN        NaN
##   7     0e+00        NaN        NaN        NaN        NaN
##   7     1e-04        NaN        NaN        NaN        NaN
##   7     1e-03        NaN        NaN        NaN        NaN
##   7     1e-02        NaN        NaN        NaN        NaN
##   7     1e-01        NaN        NaN        NaN        NaN
##   9     0e+00        NaN        NaN        NaN        NaN
##   9     1e-04        NaN        NaN        NaN        NaN
##   9     1e-03        NaN        NaN        NaN        NaN
##   9     1e-02        NaN        NaN        NaN        NaN
##   9     1e-01        NaN        NaN        NaN        NaN
##
## F was used to select the optimal model using the largest value.
## The final values used for the model were size = 3 and decay = 0.1.
```

```
NN1v <- predict(NN1, val_m)
confusionMatrix(NN1v, clean_val_type, mode = "prec_recall", positive = "spam")
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction spam ham
##       spam  314  20
##       ham    38 575
##
##                Accuracy : 0.9388
##                  95% CI : (0.9215, 0.9532)
##     No Information Rate : 0.6283
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8675
##
##  Mcnemar's Test P-Value : 0.0256
##
##               Precision : 0.9401
##                  Recall : 0.8920
##                      F1 : 0.9155
##              Prevalence : 0.3717
##          Detection Rate : 0.3316
##    Detection Prevalence : 0.3527
##       Balanced Accuracy : 0.9292
##
##        'Positive' Class : spam
##
```

Some light tuning:

```r
nn_grid <- expand.grid(size = 1L, decay = c(0.99, seq(0.95, 0.05, -0.05), 0.01))
NN2 <- train(x = train_m, y = clean_train_type, method = "nnet", trace = FALSE,
             trControl = tr_ctrl, metric = "F", tuneGrid = nn_grid,
             maxit = 250L)
NN2
```

```
## Neural Network
##
## 1943 samples
##  308 predictor
##    2 classes: 'spam', 'ham'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1749, 1748, 1748, 1748, 1750, 1749, ...
## Resampling results across tuning parameters:
##
##   decay  AUC        Precision  Recall     F
##   0.01   0.6383898  0.9212195  0.9121946  0.9162207
##   0.05   0.7692994  0.9240296  0.9265839  0.9247392
##   0.10   0.8072211  0.9355148  0.9208489  0.9274281
##   0.15   0.8402565  0.9353433  0.9208282  0.9274672
##   0.20   0.8527932  0.9347728  0.9293996  0.9313668
##   0.25   0.8140667  0.9313337  0.9308282  0.9303504
##   0.30   0.8663416  0.9369526  0.9208075  0.9279617
```

```
##   0.35   0.8758571  0.9408470  0.9308282  0.9349326
##   0.40   0.8753815  0.9421074  0.9294203  0.9349393
##   0.45   0.8786946  0.9405514  0.9294410  0.9341948
##   0.50   0.8834925  0.9433573  0.9308075  0.9362327
##   0.55   0.8856740  0.9518686  0.9265010  0.9380507
##   0.60   0.8848396  0.9489563  0.9279503  0.9374756
##   0.65   0.8895250  0.9503670  0.9322567  0.9404662
##   0.70   0.8917974  0.9462910  0.9293996  0.9369784
##   0.75   0.8945690  0.9476758  0.9279503  0.9367894
##   0.80   0.9030283  0.9518262  0.9250932  0.9372766
##   0.85   0.9035252  0.9531726  0.9294203  0.9402971
##   0.90   0.9059117  0.9528485  0.9265217  0.9386527
##   0.95   0.9077766  0.9539975  0.9279503  0.9400688
##   0.99   0.9121959  0.9570959  0.9250725  0.9398958
##
## Tuning parameter 'size' was held constant at a value of 1
## F was used to select the optimal model using the largest value.
## The final values used for the model were size = 1 and decay = 0.65.
```

```r
NN2v <- predict(NN2, val_m)
confusionMatrix(NN2v, clean_val_type, mode = "prec_recall", positive = "spam")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction spam ham
##       spam  320  15
##       ham    32 580
##
##                Accuracy : 0.9504
##                  95% CI : (0.9345, 0.9633)
##     No Information Rate : 0.6283
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8927
##
##  Mcnemar's Test P-Value : 0.0196
##
##               Precision : 0.9552
##                  Recall : 0.9091
##                      F1 : 0.9316
##              Prevalence : 0.3717
##          Detection Rate : 0.3379
##    Detection Prevalence : 0.3537
##       Balanced Accuracy : 0.9419
##
##        'Positive' Class : spam
##
```

Both models performed the same on the validation set. As the second performed better on the training set too, we will use it.

Feature importance

Which terms had the most influence on ham/spam classification using a Neural Network?

```r
# estimate variable importance
importance <- varImp(NN2)
# summarize importance
print(importance)
```

```
## nnet variable importance
##
##   only 20 most important variables shown (out of 308)
##
##                      Overall
## url                   100.00
## click                  85.11
## wrote                  69.95
## visit                  57.65
## write                  46.26
## guarante               43.72
## use                    43.49
## seem                   40.65
## satalk                 39.19
## old                    39.14
## two                    38.81
## home                   38.03
## futur                  35.39
## dollar                 35.37
## file                   35.15
## credit                 35.08
## contenttyp             33.92
## minut                  33.17
## repli                  33.17
## contenttransferencod   32.85
```

Gradient Boosted Machines

```r
tr_ctrl <- trainControl(method = "cv", number = 10L, classProbs = TRUE,
                        summaryFunction = prSummary)
GBM1 <- train(x = train_m, y = clean_train_type, method = "gbm", verbose = FALSE,
              trControl = tr_ctrl, tuneLength = 5L, metric = "F")
GBM1v <- predict(GBM1, val_m)
confusionMatrix(GBM1v, clean_val_type, mode = "prec_recall", positive = "spam")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction spam ham
##       spam  325  11
##       ham    27 584
##
##                Accuracy : 0.9599
##                  95% CI : (0.9453, 0.9715)
##     No Information Rate : 0.6283
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9133
##
##  Mcnemar's Test P-Value : 0.01496
##
##               Precision : 0.9673
##                  Recall : 0.9233
##                      F1 : 0.9448
##              Prevalence : 0.3717
##          Detection Rate : 0.3432
##    Detection Prevalence : 0.3548
##       Balanced Accuracy : 0.9524
##
##        'Positive' Class : spam
##
```

This model looks really good. Let's throw a little extra fine-tuning in. After running a wide-scale grid, the best option is selected below, so that the entire grid doesn't have to rerun every time.

```r
gbm_grid <- expand.grid(n.trees = 400L,
                     interaction.depth = 7L,
                     shrinkage = 0.1,
                     n.minobsinnode = 10L)
GBM2 <- train(x = train_m, y = clean_train_type, method = "gbm", verbose = FALSE,
              trControl = tr_ctrl, tuneGrid = gbm_grid, metric = "F")
GBM2
```

```
## Stochastic Gradient Boosting
##
## 1943 samples
##  308 predictor
##    2 classes: 'spam', 'ham'
```

```
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1749, 1750, 1748, 1748, 1749, 1748, ...
## Resampling results:
##
##   AUC        Precision  Recall     F
##   0.9716112  0.9437718  0.9294824  0.936128
##
## Tuning parameter 'n.trees' was held constant at a value of 400
##  7
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
```

```r
GBM2v <- predict(GBM2, val_m)
confusionMatrix(GBM2v, clean_val_type, mode = "prec_recall", positive = "spam")
```

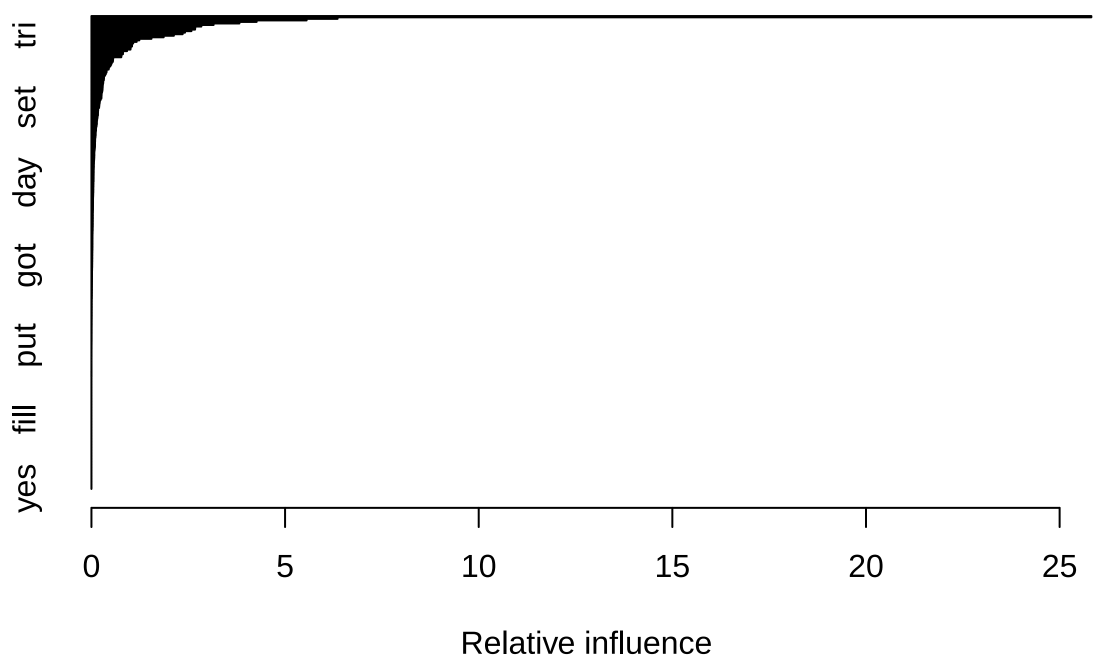
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction spam ham
##       spam  325  12
##       ham    27 583
##
##                Accuracy : 0.9588
##                  95% CI : (0.9441, 0.9706)
##     No Information Rate : 0.6283
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9111
##
##  Mcnemar's Test P-Value : 0.02497
##
##               Precision : 0.9644
##                  Recall : 0.9233
##                      F1 : 0.9434
##              Prevalence : 0.3717
##          Detection Rate : 0.3432
##    Detection Prevalence : 0.3559
##       Balanced Accuracy : 0.9516
##
##        'Positive' Class : spam
##
```

The second model performed better.

Feature importance

Which terms had the most influence on ham/spam classification using a Gradient Boosted Machines?

```r
# estimate variable importance
summary(GBM2)
```




```
##                                      var      rel.inf
## click                              click 2.581944e+01
## wrote                              wrote 6.355481e+00
## contenttransferencod contenttransferencod 5.556169e+00
## url                                  url 4.265440e+00
## email                              email 3.820689e+00
## credit                            credit 3.154606e+00
## inform                            inform 2.838570e+00
## free                                free 2.681326e+00
## repli                              repli 2.675613e+00
## visit                              visit 2.581664e+00
## use                                  use 2.415561e+00
## receiv                            receiv 2.355206e+00
## tri                                  tri 2.128315e+00
## remov                              remov 1.865354e+00
## will                                will 1.550087e+00
## satalk                            satalk 1.236813e+00
## dollar                            dollar 1.167111e+00
## said                                said 1.078139e+00
```

```
## spam                                          spam 1.050712e+00
## money                                        money 1.047632e+00
## think                                        think 1.015805e+00
## guarante                                  guarante 1.009096e+00
## life                                          life 9.194919e-01
## onlin                                        onlin 8.165683e-01
## file                                          file 8.138042e-01
## write                                        write 7.783694e-01
## home                                          home 7.707283e-01
## month                                        month 5.625751e-01
## seem                                          seem 5.579588e-01
## internet                                  internet 5.535825e-01
## busi                                          busi 5.251096e-01
## origin                                      origin 5.096281e-01
## run                                            run 4.878476e-01
## price                                        price 4.530023e-01
## base                                          base 4.445244e-01
## offer                                        offer 3.944501e-01
## still                                        still 3.892529e-01
## opportun                                  opportun 3.714325e-01
## old                                            old 3.534826e-01
## someth                                      someth 3.247006e-01
## sure                                          sure 3.236444e-01
## site                                          site 3.202524e-01
## like                                          like 3.048405e-01
## compani                                    compani 3.038267e-01
## list                                          list 2.964294e-01
## contact                                    contact 2.908383e-01
## multipart                                multipart 2.898901e-01
## get                                            get 2.865733e-01
## increas                                    increas 2.828737e-01
## order                                        order 2.742940e-01
## market                                      market 2.639727e-01
## linux                                        linux 2.616707e-01
## two                                            two 2.592182e-01
## find                                          find 2.564386e-01
## messag                                      messag 2.341058e-01
## first                                        first 2.168344e-01
## say                                            say 2.124561e-01
## minut                                        minut 2.054213e-01
## window                                      window 2.013475e-01
## set                                            set 1.978504e-01
## welcom                                      welcom 1.730781e-01
## thank                                        thank 1.727624e-01
## date                                          date 1.688326e-01
## mail                                          mail 1.677053e-01
## group                                        group 1.676549e-01
## sinc                                          sinc 1.576445e-01
## post                                          post 1.530346e-01
## can                                            can 1.456841e-01
## new                                            new 1.402387e-01
## problem                                    problem 1.393742e-01
## fix                                            fix 1.381567e-01
## sponsor                                    sponsor 1.314840e-01
```

```
## help                                   help 1.228502e-01
## page                                   page 1.180051e-01
## anyth                                 anyth 1.178928e-01
## place                                 place 1.105133e-01
## develop                             develop 1.096654e-01
## come                                   come 1.087585e-01
## just                                   just 1.047112e-01
## send                                   send 9.805449e-02
## time                                   time 9.485107e-02
## form                                   form 9.406089e-02
## mime                                   mime 9.223432e-02
## know                                   know 9.208486e-02
## thought                             thought 9.144945e-02
## possibl                             possibl 8.965919e-02
## detail                               detail 8.452459e-02
## per                                     per 8.131073e-02
## world                                 world 7.396026e-02
## open                                   open 7.388642e-02
## mani                                   mani 7.351233e-02
## now                                     now 7.288770e-02
## call                                   call 7.099063e-02
## end                                     end 6.974839e-02
## futur                                 futur 6.546688e-02
## forward                             forward 6.434879e-02
## keep                                   keep 6.164405e-02
## user                                   user 6.160194e-02
## buy                                     buy 6.147439e-02
## exampl                               exampl 5.950607e-02
## big                                     big 5.805943e-02
## hour                                   hour 5.756405e-02
## anyon                                 anyon 5.704610e-02
## friend                               friend 5.655791e-02
## recent                               recent 5.590497e-02
## result                               result 5.566329e-02
## peopl                                 peopl 5.535915e-02
## thing                                 thing 5.489840e-02
## day                                     day 5.030074e-02
## includ                               includ 5.005320e-02
## found                                 found 4.991421e-02
## understand                       understand 4.791879e-02
## sfnet                                 sfnet 4.759896e-02
## custom                               custom 4.727151e-02
## secur                                 secur 4.692483e-02
## test                                   test 4.516230e-02
## look                                   look 4.438891e-02
## start                                 start 4.370214e-02
## around                               around 4.131996e-02
## cours                                 cours 4.080613e-02
## septemb                             septemb 4.063863e-02
## bit                                     bit 4.039984e-02
## last                                   last 3.999599e-02
## high                                   high 3.993277e-02
## person                               person 3.947837e-02
## address                             address 3.925785e-02
```

```
## mean                                   mean 3.799129e-02
## comput                               comput 3.633440e-02
## next                                   next 3.527726e-02
## one                                     one 3.512288e-02
## info                                   info 3.492654e-02
## import                               import 3.490083e-02
## wed                                     wed 3.473488e-02
## seen                                   seen 3.454723e-02
## want                                   want 3.405116e-02
## much                                   much 3.385459e-02
## better                               better 3.350655e-02
## today                                 today 3.202067e-02
## differ                               differ 3.152534e-02
## updat                                 updat 3.138940e-02
## probabl                             probabl 2.917570e-02
## million                             million 2.883686e-02
## read                                   read 2.654960e-02
## phone                                 phone 2.631754e-02
## rate                                   rate 2.592384e-02
## may                                     may 2.557043e-02
## real                                   real 2.554987e-02
## bythinkgeek                     bythinkgeek 2.548366e-02
## point                                 point 2.543418e-02
## instal                               instal 2.490238e-02
## back                                   back 2.452754e-02
## simpli                               simpli 2.412530e-02
## power                                 power 2.340569e-02
## access                               access 2.209833e-02
## contenttyp                       contenttyp 2.106975e-02
## ever                                   ever 2.102390e-02
## etc                                     etc 2.097603e-02
## return                               return 2.032764e-02
## build                                 build 2.021526e-02
## provid                               provid 1.986608e-02
## code                                   code 1.961677e-02
## els                                     els 1.877471e-02
## got                                     got 1.829520e-02
## someon                               someon 1.810548e-02
## noth                                   noth 1.646334e-02
## alreadi                             alreadi 1.561694e-02
## stuff                                 stuff 1.456678e-02
## word                                   word 1.451537e-02
## total                                 total 1.439143e-02
## begin                                 begin 1.433293e-02
## process                             process 1.401823e-02
## show                                   show 1.361424e-02
## data                                   data 1.303904e-02
## case                                   case 1.295910e-02
## talk                                   talk 1.262082e-02
## past                                   past 1.221493e-02
## complet                             complet 1.195687e-02
## number                               number 1.180451e-02
## servic                               servic 1.162272e-02
## make                                   make 1.133074e-02
```

```
## let                                   let 1.075925e-02
## year                                 year 1.034680e-02
## wait                                 wait 1.026133e-02
## version                           version 9.528826e-03
## week                                 week 8.520263e-03
## work                                 work 7.905414e-03
## avail                               avail 7.711123e-03
## geek                                 geek 7.558132e-03
## network                           network 7.460938e-03
## see                                   see 7.363169e-03
## news                                 news 7.239714e-03
## subject                           subject 6.985278e-03
## regard                             regard 6.981816e-03
## direct                             direct 6.921498e-03
## idea                                 idea 6.803892e-03
## good                                 good 6.762419e-03
## web                                   web 6.387431e-03
## communic                         communic 5.860654e-03
## account                           account 5.816220e-03
## bill                                 bill 5.617547e-03
## part                                 part 5.337452e-03
## server                             server 5.316743e-03
## program                           program 4.960489e-03
## take                                 take 4.778376e-03
## either                             either 4.736612e-03
## packag                             packag 4.706863e-03
## easi                                 easi 4.676668e-03
## special                           special 4.391545e-03
## also                                 also 4.256476e-03
## question                         question 4.252827e-03
## save                                 save 4.231435e-03
## actual                             actual 4.042317e-03
## kind                                 kind 3.437196e-03
## live                                 live 3.339878e-03
## put                                   put 2.728495e-03
## realli                             realli 2.620236e-03
## product                           product 2.523956e-03
## interest                         interest 2.045900e-03
## within                             within 2.038240e-03
## without                           without 1.990595e-03
## request                           request 1.900643e-03
## report                             report 1.801369e-03
## give                                 give 1.685072e-03
## issu                                 issu 1.593677e-03
## simpl                               simpl 1.575688e-03
## format                             format 1.560153e-03
## pay                                   pay 1.535011e-03
## experi                             experi 1.478700e-03
## lot                                   lot 1.363120e-03
## manag                               manag 1.222567e-03
## learn                               learn 1.217741e-03
## name                                 name 8.338569e-04
## believ                             believ 7.771256e-04
## heaven                             heaven 7.048928e-04
```

```
## sent                                   sent 5.812477e-04
## abl                                     abl 0.000000e+00
## accept                               accept 0.000000e+00
## add                                     add 0.000000e+00
## allow                                 allow 0.000000e+00
## alway                                 alway 0.000000e+00
## anoth                                 anoth 0.000000e+00
## ask                                     ask 0.000000e+00
## aug                                     aug 0.000000e+00
## bad                                     bad 0.000000e+00
## best                                   best 0.000000e+00
## box                                     box 0.000000e+00
## chang                                 chang 0.000000e+00
## check                                 check 0.000000e+00
## cost                                   cost 0.000000e+00
## countri                             countri 0.000000e+00
## creat                                 creat 0.000000e+00
## current                             current 0.000000e+00
## done                                   done 0.000000e+00
## effect                               effect 0.000000e+00
## enough                               enough 0.000000e+00
## error                                 error 0.000000e+00
## even                                   even 0.000000e+00
## everi                                 everi 0.000000e+00
## everyth                             everyth 0.000000e+00
## fact                                   fact 0.000000e+00
## fast                                   fast 0.000000e+00
## feel                                   feel 0.000000e+00
## fill                                   fill 0.000000e+00
## follow                               follow 0.000000e+00
## full                                   full 0.000000e+00
## great                                 great 0.000000e+00
## happen                               happen 0.000000e+00
## hope                                   hope 0.000000e+00
## howev                                 howev 0.000000e+00
## instead                             instead 0.000000e+00
## least                                 least 0.000000e+00
## less                                   less 0.000000e+00
## limit                                 limit 0.000000e+00
## line                                   line 0.000000e+00
## link                                   link 0.000000e+00
## long                                   long 0.000000e+00
## made                                   made 0.000000e+00
## mayb                                   mayb 0.000000e+00
## must                                   must 0.000000e+00
## need                                   need 0.000000e+00
## never                                 never 0.000000e+00
## note                                   note 0.000000e+00
## profession                       profession 0.000000e+00
## quick                                 quick 0.000000e+00
## reason                               reason 0.000000e+00
## relat                                 relat 0.000000e+00
## releas                               releas 0.000000e+00
## requir                               requir 0.000000e+00
```

```
## right                                    right 0.000000e+00
## rpmlist                                rpmlist 0.000000e+00
## second                                  second 0.000000e+00
## sell                                      sell 0.000000e+00
## sep                                        sep 0.000000e+00
## sign                                      sign 0.000000e+00
## softwar                                softwar 0.000000e+00
## sourc                                    sourc 0.000000e+00
## state                                    state 0.000000e+00
## support                                support 0.000000e+00
## system                                  system 0.000000e+00
## tell                                      tell 0.000000e+00
## textplain                            textplain 0.000000e+00
## though                                  though 0.000000e+00
## type                                      type 0.000000e+00
## unsubscrib                          unsubscrib 0.000000e+00
## way                                        way 0.000000e+00
## well                                      well 0.000000e+00
## wish                                      wish 0.000000e+00
## yes                                        yes 0.000000e+00
```

Other models

With over 230 possible models, there are many more options to train, like XGBoost, Neural Networks, Bayesian Regression, Support Vector Machines, etc. We don't need to exhaust the possibilities here.

Test Models

The best models in the above categories will now be compared against the testing/holdout set:

```
LogRt <- predict(LogR1, test_m)
RFt <- predict(RF1, newdata=test_m)
NNt <- predict(NN2, test_m)
NBt <- predict(NB2, test_m) # For laughs
GBMt <- predict(GBM2, test_m)
confusionMatrix(LogRt, clean_test_type, mode = "prec_recall", positive = "spam")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction spam ham
##       spam  323  54
##       ham    26 603
##
##                Accuracy : 0.9205
##                  95% CI : (0.902, 0.9364)
##     No Information Rate : 0.6531
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8277
##
##  Mcnemar's Test P-Value : 0.002539
##
##               Precision : 0.8568
##                  Recall : 0.9255
##                      F1 : 0.8898
##              Prevalence : 0.3469
##          Detection Rate : 0.3211
##    Detection Prevalence : 0.3748
##       Balanced Accuracy : 0.9217
##
##        'Positive' Class : spam
##
```

```
confusionMatrix(RFt, clean_test_type, mode = "prec_recall", positive = "spam")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction spam ham
##       spam  318  23
##       ham    31 634
##
##                Accuracy : 0.9463
##                  95% CI : (0.9305, 0.9594)
##     No Information Rate : 0.6531
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8809
```

```
##
##  Mcnemar's Test P-Value : 0.3408
##
##               Precision : 0.9326
##                  Recall : 0.9112
##                      F1 : 0.9217
##              Prevalence : 0.3469
##          Detection Rate : 0.3161
##    Detection Prevalence : 0.3390
##       Balanced Accuracy : 0.9381
##
##        'Positive' Class : spam
##
```

```
confusionMatrix(NNt, clean_test_type, mode = "prec_recall", positive = "spam")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction spam ham
##       spam  320  18
##       ham    29 639
##
##                Accuracy : 0.9533
##                  95% CI : (0.9384, 0.9655)
##     No Information Rate : 0.6531
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8961
##
##  Mcnemar's Test P-Value : 0.1447
##
##               Precision : 0.9467
##                  Recall : 0.9169
##                      F1 : 0.9316
##              Prevalence : 0.3469
##          Detection Rate : 0.3181
##    Detection Prevalence : 0.3360
##       Balanced Accuracy : 0.9448
##
##        'Positive' Class : spam
##
```

```
confusionMatrix(NBt, clean_test_type, mode = "prec_recall", positive = "spam")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction spam ham
##       spam   50   8
##       ham   299 649
##
##                Accuracy : 0.6948
```

```
##                  95% CI : (0.6653, 0.7232)
##     No Information Rate : 0.6531
##     P-Value [Acc > NIR] : 0.002776
##
##                   Kappa : 0.1629
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##               Precision : 0.86207
##                  Recall : 0.14327
##                      F1 : 0.24570
##              Prevalence : 0.34692
##          Detection Rate : 0.04970
##    Detection Prevalence : 0.05765
##       Balanced Accuracy : 0.56554
##
##        'Positive' Class : spam
##
```

```r
confusionMatrix(GBMt, clean_test_type, mode = "prec_recall", positive = "spam")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction spam ham
##       spam  316  18
##       ham    33 639
##
##                Accuracy : 0.9493
##                  95% CI : (0.9339, 0.962)
##     No Information Rate : 0.6531
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.887
##
##  Mcnemar's Test P-Value : 0.04995
##
##               Precision : 0.9461
##                  Recall : 0.9054
##                      F1 : 0.9253
##              Prevalence : 0.3469
##          Detection Rate : 0.3141
##    Detection Prevalence : 0.3320
##       Balanced Accuracy : 0.9390
##
##        'Positive' Class : spam
##
```

Looking across all models, Naive Bayes performed quite poorly while the remaining models all did quite well, but the winner is the **gradient boosted** model, with the highest F-score and fewest miscategorized emails of any type.

Discussion

With our initial pass on this project, we did NOT remove HTML from email messages and as a consequence, HTML tags and attribute names and values became "words" or "terms" used by our models to help resolve SPAM vs HAM. Interestingly, our models performed significantly better and the HTML terms and attribute ended up being the most important features used as criteria by models. After seeing this, we modified our email cleaning to actively remove HTML markup. Our model perform dropped ~7% across all models without HTML. This suggests that the very presense of HTML markup in the corpus is a feature associated with and predictive of SPAM.

The email corpus is from the early 2000's at a time when most email clients did NOT use HTML markup by default, so most HAM would *NOT* have included much if any HTML. SPAM on the other hand often included HTML links and images intended to draw the recipient to a website or email address where they could buy something.

While the presense of HTML was an indicator of SPAM in the early 2000's, we suspect that models trained with HTML would perform poorly today as most email clients routinely use HTML markup for text formatting, shared links and images. For this reason, we chose to remove the HTML and try training a model on only the email text, as that might perform better over time.

Note that whlie we tried to remove HTML markup, when we inspect the terms, we still see some words that look suspiciouly like HTML, for example, "contenttype". This may suggest some leakage of HTML that we missed during scrubbing.

If you inspect the terms, you may note missing trailing characters. This is not a bug, but rather part of the word stem approach to simplifying the word list by finding similar words. For example, "run", "running", "runs", "runner" all have the same base "run". The SnowballC package drops the endings so all the variants collapse to the same word root.

If we really wanted to expand this project, some additional features we might include beyond the word list:

- Possibly add a boolean feature indicating whether the email contained any URL's
- Possibly add a boolean feature whether there were any HTML markup in the email
- Use Correlation matrices to identify auto-correlation between words and remove unnecessary terms.

Since email language and markup changes over time, and spammers are constantly changing their email to get past spam filters, any model built to separate HAM vs SPAM will probably need to be constantly retrained.

Epilogue

```r
sessionInfo()
```

```
## R version 3.6.1 (2019-07-05)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 18362)
##
## Matrix products: default
##
## locale:
## [1] C
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] forcats_0.4.0      stringr_1.4.0      dplyr_0.8.3
##  [4] purrr_0.3.2        readr_1.3.1        tidyr_1.0.0
##  [7] tibble_2.1.3       tidyverse_1.2.1    wordcloud_2.6
## [10] RColorBrewer_1.1-2 caret_6.0-84       ggplot2_3.2.1
## [13] lattice_0.20-38    SnowballC_0.6.0    tm_0.7-6
## [16] NLP_0.2-0          doParallel_1.0.15  iterators_1.0.12
## [19] foreach_1.4.7
##
## loaded via a namespace (and not attached):
##  [1] httr_1.4.1         jsonlite_1.6       splines_3.6.1
##  [4] prodlim_2018.04.18 modelr_0.1.5       assertthat_0.2.1
##  [7] highr_0.8          stats4_3.6.1       cellranger_1.1.0
## [10] yaml_2.2.0         slam_0.1-46        ipred_0.9-9
## [13] pillar_1.4.2       backports_1.1.5    glue_1.3.1
## [16] digest_0.6.20      rvest_0.3.4        colorspace_1.4-1
## [19] recipes_0.1.7      htmltools_0.4.0    Matrix_1.2-17
## [22] plyr_1.8.4         timeDate_3043.102  pkgconfig_2.0.3
## [25] broom_0.5.2        haven_2.1.1        scales_1.0.0
## [28] gower_0.2.1        lava_1.6.6         generics_0.0.2
## [31] withr_2.1.2        nnet_7.3-12        lazyeval_0.2.2
## [34] cli_1.1.0          survival_2.44-1.1  magrittr_1.5
## [37] crayon_1.3.4       readxl_1.3.1       evaluate_0.14
## [40] nlme_3.1-141       MASS_7.3-51.4      xml2_1.2.2
## [43] class_7.3-15       tools_3.6.1        data.table_1.12.2
## [46] hms_0.5.2          lifecycle_0.1.0    munsell_0.5.0
## [49] compiler_3.6.1     rlang_0.4.0        grid_3.6.1
## [52] rstudioapi_0.10    labeling_0.3       rmarkdown_1.16
## [55] gtable_0.3.0       ModelMetrics_1.2.2 codetools_0.2-16
## [58] reshape2_1.4.3     R6_2.4.0           lubridate_1.7.4
## [61] knitr_1.25         zeallot_0.1.0      stringi_1.4.3
## [64] Rcpp_1.0.2         vctrs_0.2.0        rpart_4.1-15
## [67] tidyselect_0.2.5   xfun_0.10
```

```r
stopCluster(cl)
```