Homework 9

Linh Vu

Instructions

1. Add your name between the quotations marks on the author line in the YAML above.
2. Work through the problems below, composing your answer to each question between the bars of red stars.
3. Be sure to make commits often and push your commits to Github periodically.
4. When finished, knit your document to .pdf and view the resulting output to ensure it matches your intent.
5. To submit your assignment, make one final commit. Then make push your repo with all commits back to Github one final time.

Classification Competition

Your objective is to build a classification model **using the tools we have discussed in class** that predicts whether an email is spam, based on other characteristics of the email. You will construct your model using a training data set with information on 57 variables recorded for 919 emails. I've also included an evaluation set consisting of the predictor values (but not the responses) for 309 additional emails. You will use your models to make predictions on these additional emails, which I will compare to their true spam status (that I have recorded, but have withheld) in order to evaluate your model.

You should record your answers in this .Rmd file. However, you are encouraged to use a separate .Rmd file for scratchwork. You will be asked to reflect on your thought process, setbacks, and successes, as well as the code you included in this homework, during our second Code Review week 13, so be sure to keep detailed notes of your investigation.

The assignment is divided into several **Components** to help organize your work. Put all work you want graded between the bars of red stars in the corresponding section.

Grading

Your final score on this assignment will be based 15% on the **performance** of your model predictions on the evaluation data, as measured by accuracy, sensitivity and specificity (as detailed below) and 85% on the quality and depth of your explanations and analysis.

I *do not* expect you to find the absolute best model for this data set. In fact, it is entirely possible to earn top marks on this assignment with a model of mediocre accuracy, provided you submit insightful analysis based on topics we have investigated in our course.

You will be graded as much on your discussion of what *didn't work* and why, as what did.

Goal

At the end of this assignment, you will construct 3 models that you feel best achieve each of the following goals. I've also included benchmarks to help you assess whether you've achieved these goals:

1. The model with the highest overall accuracy (at least 90%)
2. The model with the highest specificity while maintaining reasonable accuracy (specificity at least 95%, accuracy at least 80%)
3. The model with the highest sensitivity while maintaining reasonable accuracy (sensitivity at least 95%, accuracy at least 80%)

Achieving each of these goals will earn at least a B+ on the "performance" component of the assignment.

The Data

The data set `spam_train` can be found in the hw_9 repo and can be loaded by running the following code.

```
spam_train<-read.csv("spam_train.csv")
spam_train$spam <- as.factor(spam_train$spam)
```

Additionally, the `data_description.txt` file in the same repo gives a full description of the variables appearing in the data set.

There is one special column of note:

- `spam` is your response variable (coded as a factor variable, where 1 = spam and 0 = not spam) and should not be included as a predictor.

Components

Data Exploration

In this section, you should perform preliminary data exploration and analysis. This data set is a bit too large to do a full investigation of each variable, so select several variables you think may be useful (at least 6 for visualization purposes, but you may want to use many more for the actual model building). Create visualizations for each variable individually, along with visualizations showing the relationship between the variable and the response. Compute relevant summary statistics for each of your variables.

```
# Load the necessary libraries
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
library(ISLR)
library(tidyr)
library(rsample)
library(purrr)
library(modelr)
library(yardstick)
```

```
##
## Attaching package: 'yardstick'
```

```
## The following objects are masked from 'package:modelr':
##
##     mae, mape, rmse
```

```
library(kknn)
library(e1071)
```

```
##
## Attaching package: 'e1071'
```

```
## The following object is masked from 'package:rsample':
##
##     permutations
```

```
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

To perform preliminary data exploration and analysis, we will first look at the structure of the data set:

```r
str(spam_train)
```

```
## 'data.frame':    919 obs. of  58 variables:
##  $ word_freq_make           : num  0 0 0 0 0.26 0 0 0.11 0.51 0 ...
##  $ word_freq_address        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ word_freq_al             : num  0 0 0 0.54 0 0 0.78 0.11 0.51 0.1 ...
##  $ word_freq_3d             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ word_freq_our            : num  0 0 0 0 0 0 0 0.11 0 0.1 ...
##  $ word_freq_over           : num  0 0 0 0 0 0 0 0.11 0.51 0.1 ...
##  $ word_freq_remove         : num  0 0 0 0 0 0 0 0 0 0.2 ...
##  $ word_freq_internet       : num  0 0 0 0 0 0 0 0 0 0.2 ...
##  $ word_freq_order          : num  0 0 0 0 0 0 0 1.03 0 0.1 ...
##  $ word_freq_mail           : num  0.85 0 2.32 0 0.53 0 0 0 0 0 ...
##  $ word_freq_receive        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ word_freq_will           : num  0 1.85 0 0.27 3.76 0.91 0.78 0.34 0.51 0.2 ...
##  $ word_freq_people         : num  0 0 0 0 0 0 0 0 0 0.1 ...
##  $ word_freq_report         : num  0 0 0 0 0 0 0 0 0 0.2 ...
##  $ word_freq_addresses      : num  1.7 0 0 0 0 0 0 0 0 0 ...
##  $ word_freq_free           : num  0 0 2.32 0 0 0 0 0 0 0 ...
##  $ word_freq_business       : num  0 0 0 0 0 0.91 0 0 0 0 ...
##  $ word_freq_email          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ word_freq_you            : num  2.56 0 2.32 3.29 0.26 2.75 1.56 0.45 0.51 0 ...
##  $ word_freq_credit         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ word_freq_your           : num  1.7 0 0 0 0 0 0 0.22 1.02 0 ...
##  $ word_freq_font           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ word_freq_000            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ word_freq_money          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ word_freq_hp             : num  0 0 0 0 3.76 1.83 0 0.57 0 3 ...
##  $ word_freq_hp.1           : num  0 0 0 0 2.68 0 0 0.68 0 0 ...
##  $ word_freq_george         : num  0 0 0 0 0 0 0 0.11 0 0 ...
##  $ word_freq_650            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ word_freq_lab            : num  0 1.85 0 0 0 0 0 0 0 0 ...
##  $ word_freq_labs           : num  0 0 0 0 0.26 0 0 0 0 0 ...
##  $ word_freq_telnet         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ word_freq_857            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ word_freq_data           : num  0 0 0 0 0 0 0 0.34 0 0 ...
##  $ word_freq_415            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ word_freq_85             : num  0 0 0 0 0.26 0 0 0 0 0 ...
##  $ word_freq_technology     : num  0 0 0 0 0 0 0 0 0 0.31 ...
##  $ word_freq_1999           : num  0 1.85 0 0.27 0 0.91 0 0.22 0.51 0 ...
##  $ word_freq_parts          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ word_freq_pm             : num  0 0 0 0.27 0 0 0 0 0.51 0 ...
##  $ word_freq_direct         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ word_freq_cs             : num  0 0 2.32 0.54 0 0 0 0 0.51 0 ...
##  $ word_freq_meeting        : num  0 3.7 0 0 0 0.91 0 0 0 0 ...
##  $ word_freq_original       : num  0 0 0 0.27 0 0 0 0 1.02 0 ...
##  $ word_freq_project        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ word_freq_re             : num  0 0 0 0.27 0 0 0 0 0 0 ...
##  $ word_freq_edu            : num  0 0 2.32 0.27 0 0 0 0 0.51 0 ...
##  $ word_freq_table          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ word_freq_conference     : num  0 0 0 0 0 0 0.78 0 0 0.2 ...
##  $ char_freq_.              : num  0.299 0 0 0 0 0.301 0.145 0.078 0 0.013 ...
##  $ char_freq_..1            : num  0 0.308 0 0.188 0.55 0 0 0.171 0.161 0.097 ...
```

```
##  $ char_freq_..2             : num  0 0 0 0.047 0 0 0 0.031 0.08 0 ...
##  $ char_freq_..3             : num  0.149 0 0.375 0 0.045 0.301 0.725 0 0.08 0 ...
##  $ char_freq_..4             : num  0 0 0 0 0 0 0 0.031 0 0.027 ...
##  $ char_freq_..5             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ capital_run_length_average: num  1.04 2 1.44 1.75 1.84 ...
##  $ capital_run_length_longest: int  2 11 5 12 11 8 4 41 12 22 ...
##  $ capital_run_length_total  : int  26 26 13 89 158 68 19 535 66 423 ...
##  $ spam                      : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
```

The data contains 919 observations of 58 variables. Our response variable is a binary categorical variable whose levels encoded as "0" and "1". One noteworthy fact is that all of our predictors are numerical. Thus, I would want to create `spam_num`, which is the numerical version of my response variable to compute a correlation matrix among the variables.

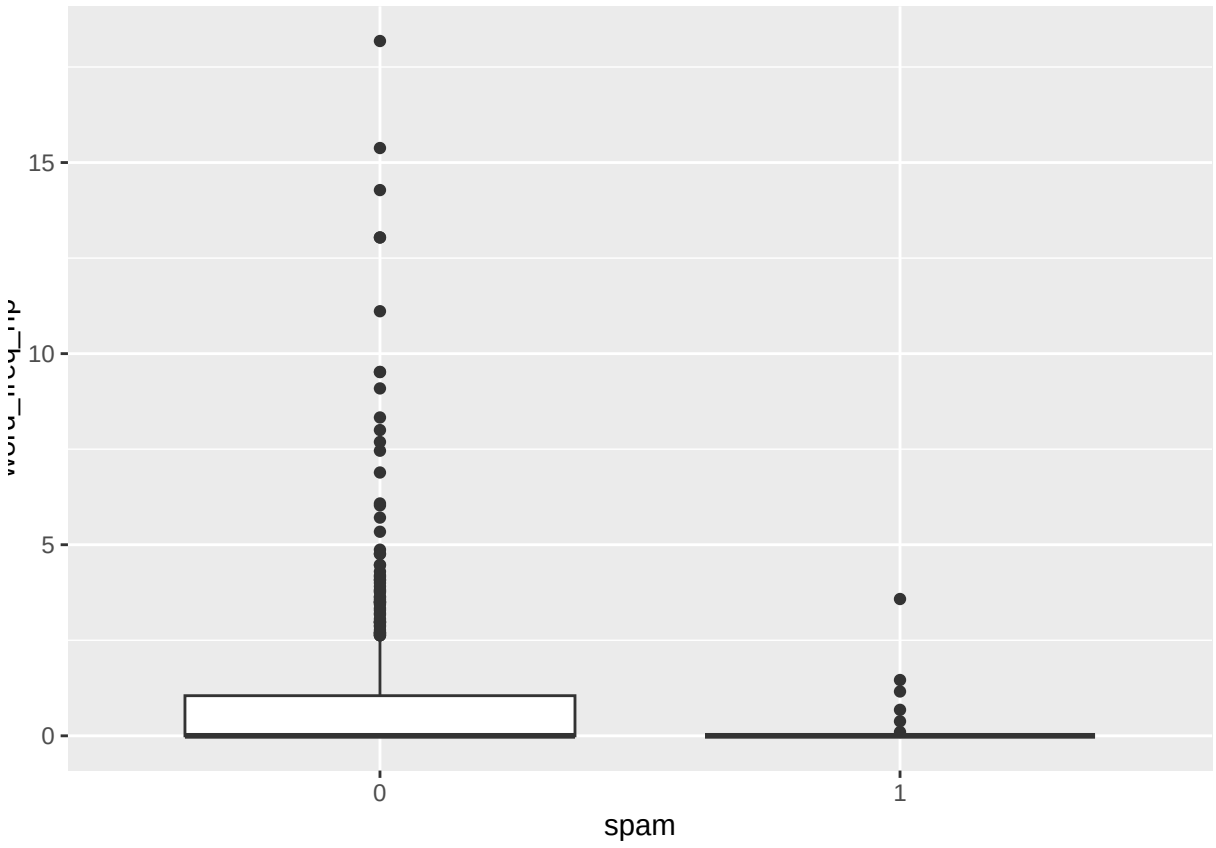
```
# Convert spam to a numerical variable to compute correlation matrix
spam_train$spam_num <- ifelse(spam_train$spam=="1", 1, 0)

# Compute correlation matrix and filter the predictors that have the strongest correlation with `spam_n
correlation <- cor(spam_train[,-58], spam_train$spam_num)
cor_dataframe <- data.frame(variable = rownames(correlation), cor = as.numeric(correlation))
cor_dataframe %>% filter(abs(cor) > 0.25)
```

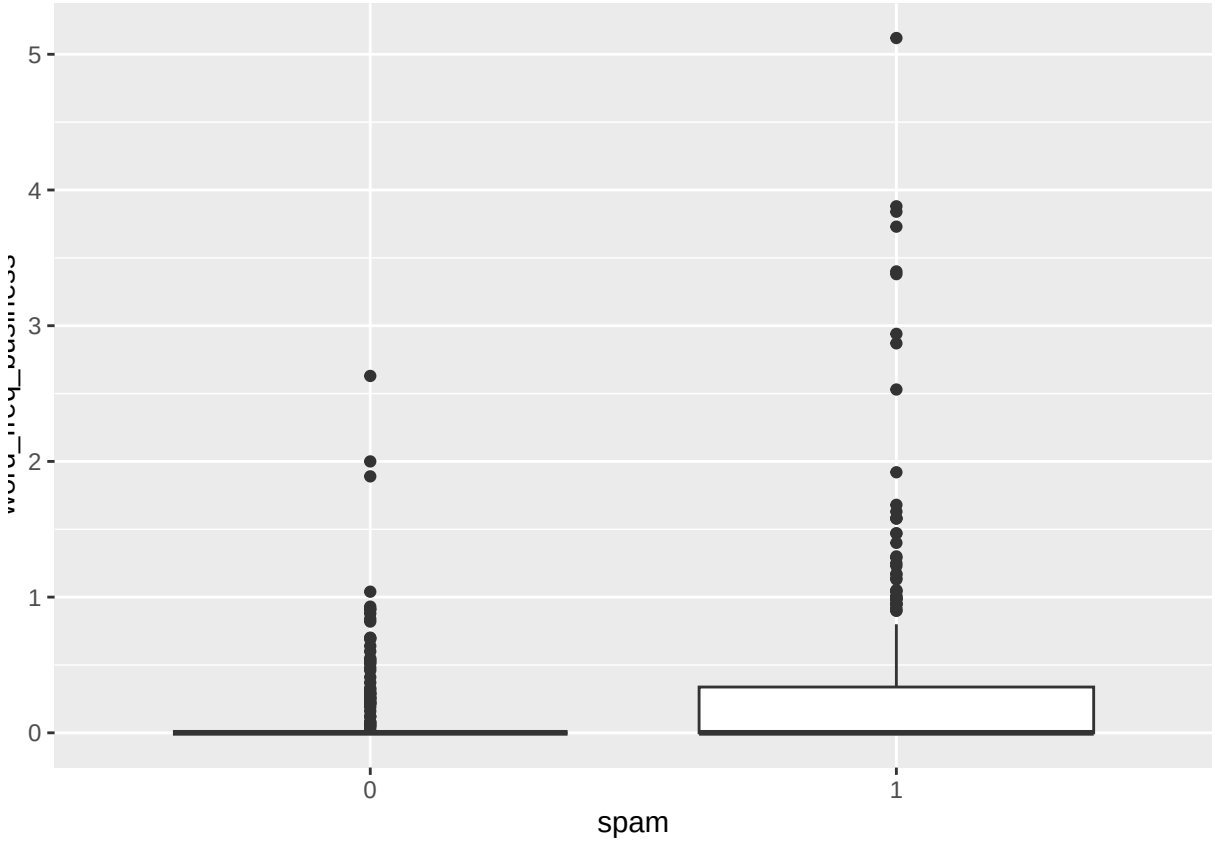
```
##                      variable        cor
## 1               word_freq_our  0.3050011
## 2              word_freq_over  0.3375928
## 3            word_freq_remove  0.3343359
## 4           word_freq_receive  0.2563150
## 5          word_freq_business  0.2670211
## 6               word_freq_you  0.3191741
## 7              word_freq_your  0.3933216
## 8               word_freq_000  0.3579312
## 9                word_freq_hp -0.2651063
## 10             word_freq_hp.1 -0.2597231
## 11              char_freq_..3  0.3119591
## 12              char_freq_..4  0.2537267
## 13 capital_run_length_longest  0.3169992
## 14                   spam_num  1.0000000
```

We would pay more attention to these predictors by creating data visualizations. I first noticed that some of these predictors have opposite correlation with the response, such as `word_freq_hp.1` (-0.2597231) and `word_freq_receive` (0.2563150), or `word_freq_hp` (-0.2651063) and `word_freq_business` (0.2670211). We can visualize this correlation through box plots:

```
# word_freq_hp is imbalanced towards spam 0
ggplot(data=spam_train) + geom_boxplot(mapping= aes(x=spam, y=word_freq_hp))
```

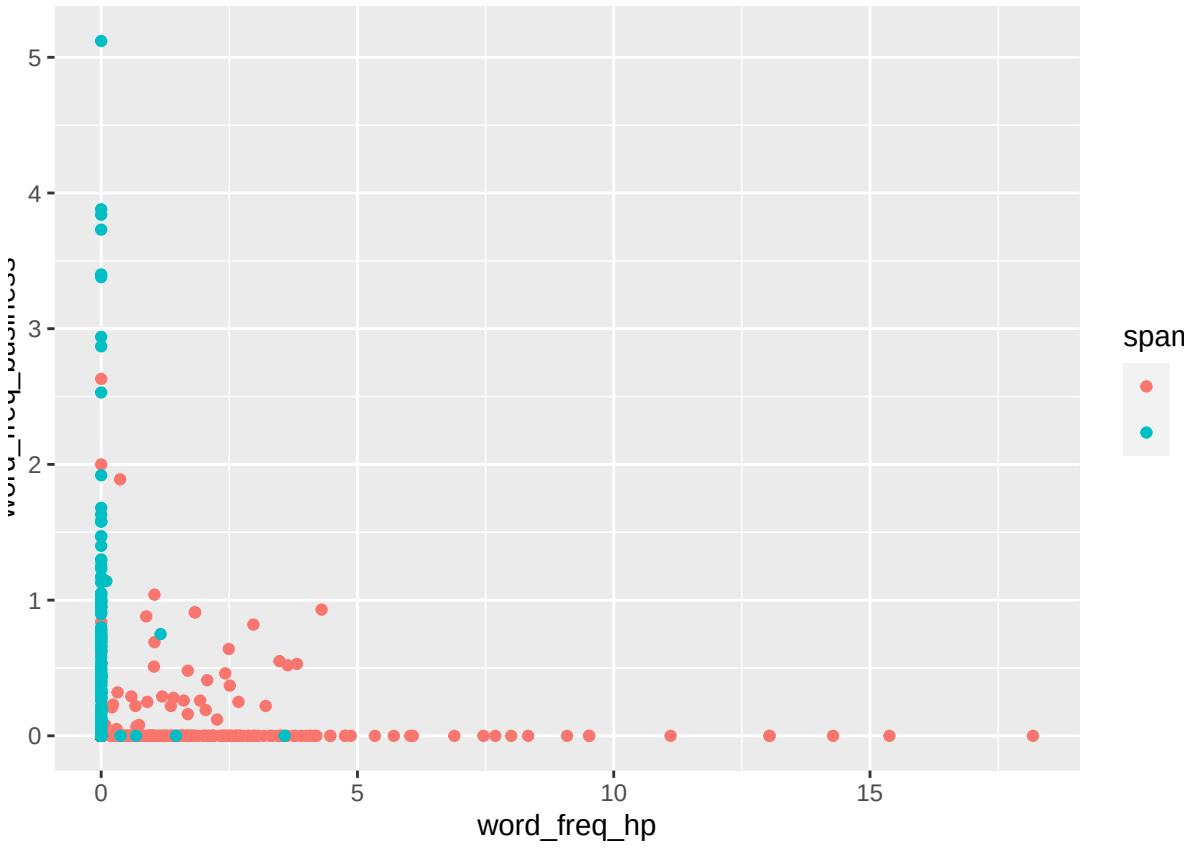



```
# word_freq_business is imbalanced towards spam 1
ggplot(data=spam_train) + geom_boxplot(mapping= aes(x=spam, y=word_freq_business))
```

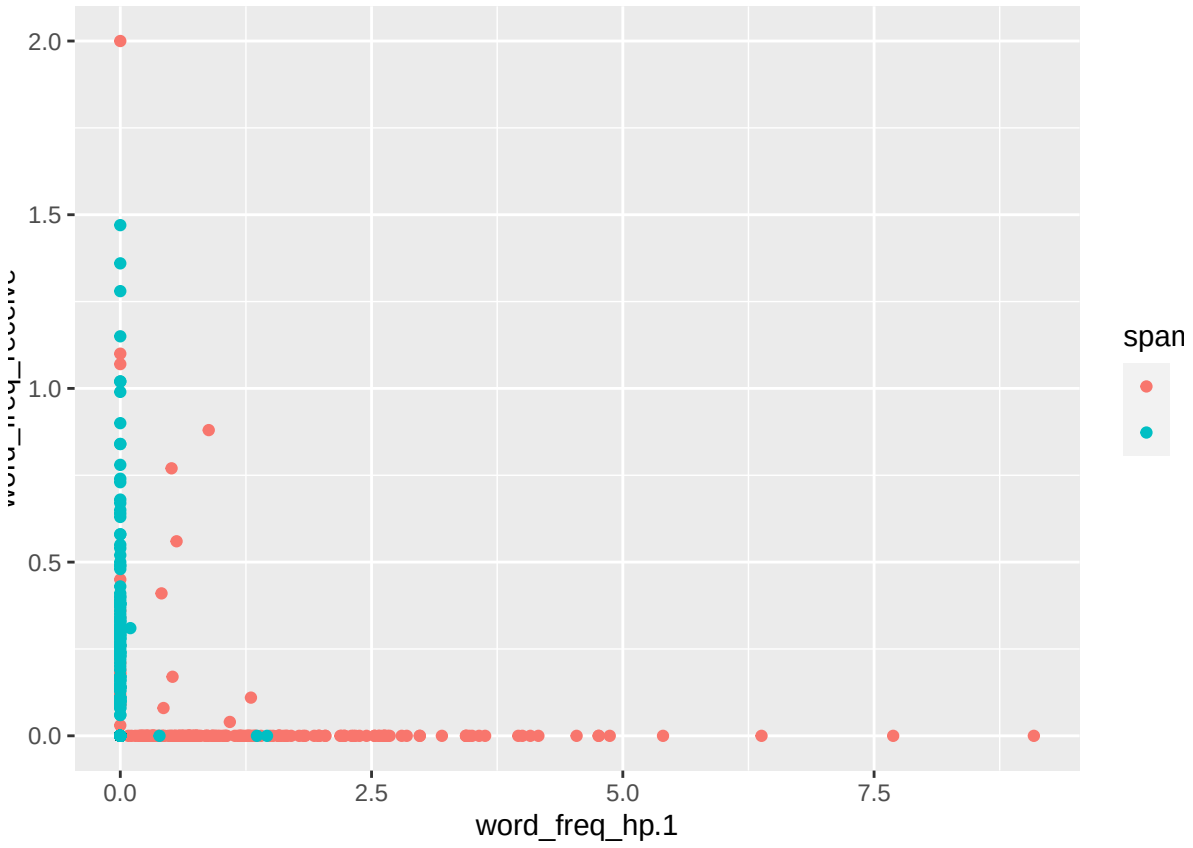


These pairs of predictors might be helpful in classifying whether or not an email is a spam. Let's confirm this belief by visualizing each pairs of predictors with opposite correlation:

```
# Scatterplot of word_freq_hp, word_freq_business, and spam coded as color
ggplot(spam_train, aes(x=word_freq_hp, y=word_freq_business)) +
geom_point(aes(color=spam))
```



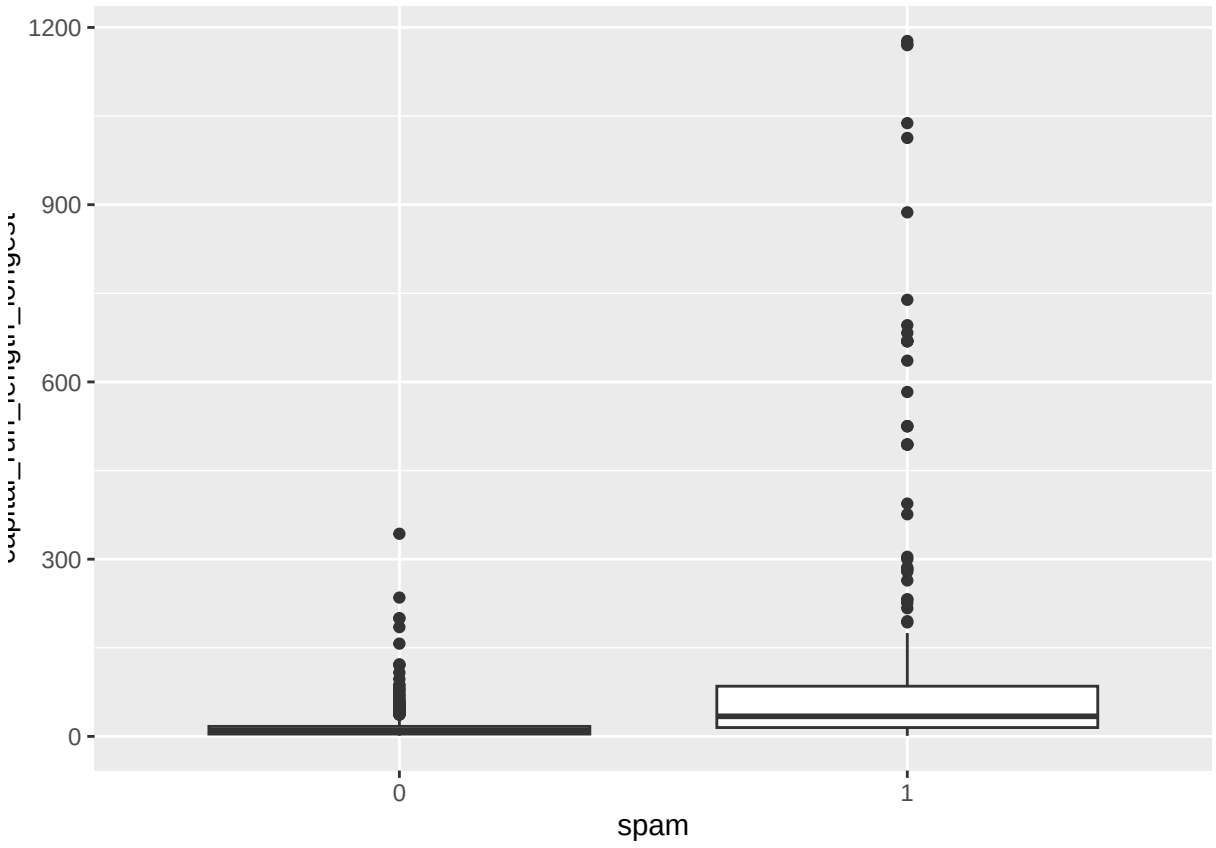

```
# Scatterplot of word_freq_hp.1, word_freq_receive, and spam coded as color
ggplot(spam_train, aes(x=word_freq_hp.1, y=word_freq_receive)) +
geom_point(aes(color=spam))
```



By knowing the values of these predictors, we would gain much information about whether or not an email is a spam.

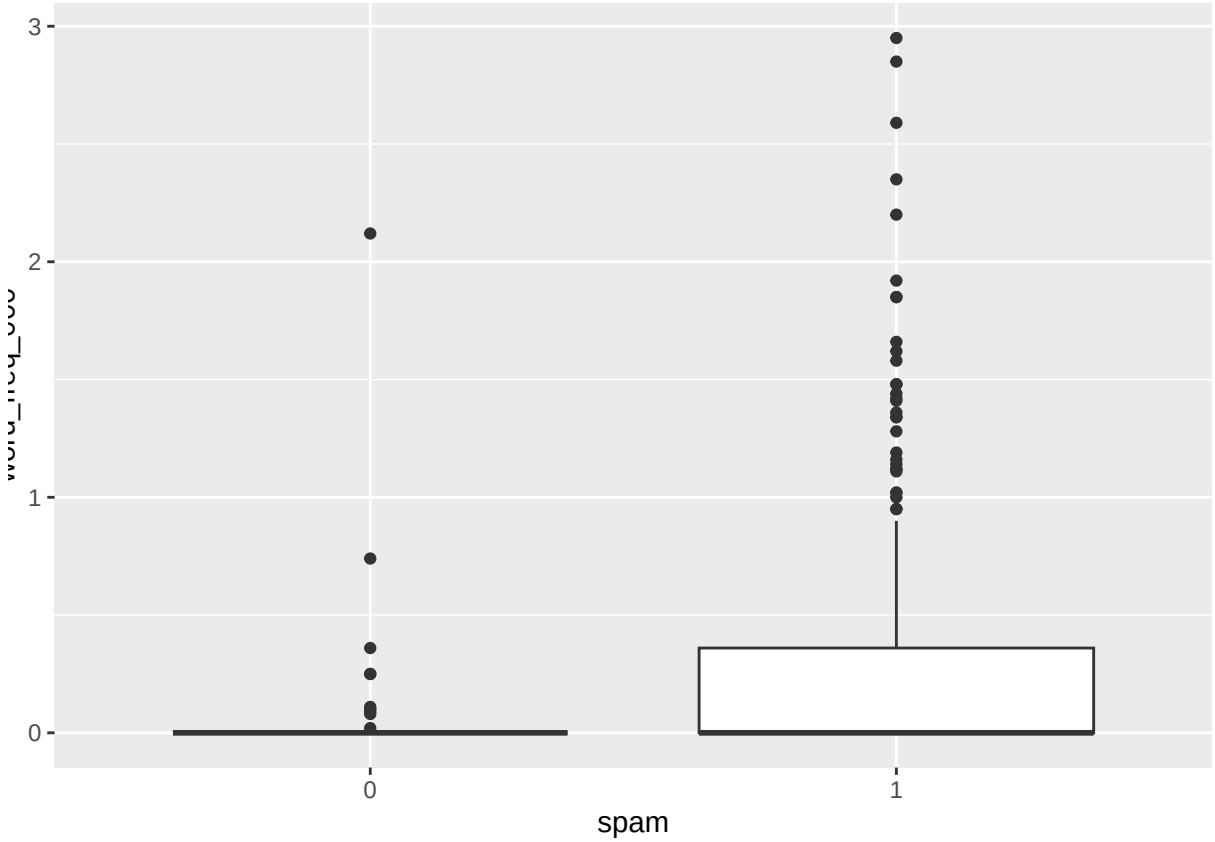
We would want to avoid predictors with too imbalanced observations among their levels, as the model might become biased towards the more frequent levels. However, there are some predictors with slightly imbalanced observations yet different range for spam and not spam emails such as `word_freq_000` and `capital_run_length_longest`.

```
# Box plot of spam and capital_run_length_longest
ggplot(data=spam_train) + geom_boxplot(mapping= aes(x=spam, y=capital_run_length_longest))
```



If the value of `capital_run_length_longest` is above 600 units, we would much more likely to classify it as a spam email.

```
# Box plot of spam and word_freq_000
ggplot(data=spam_train) + geom_boxplot(mapping= aes(x=spam, y=word_freq_000))
```



Likewise, if the value of `word_freq_000` is above 2, we would much more likely to classify it as a spam email.

Model Building

In this section, you should build a series of models (at least 5, but more is probably better) of varying complexity and that use a variety of the tools we have studied thus far. Explain why you choose to implement various features in each model. Here are some suggestions:

- Build at least one model using a large number of variables
- Build at least one model using a small number of variables
- Build at least one highly flexible model
- Build at least one highly rigid model
- Build at least one model that has a parameter that can be tuned using cross-validation
- Consider models using a variety of classification threshold (i.e. the value of c in the rule "Predict $Y = 1$ if $P(Y = 1|X) > c$")

Before building models, I would want to create splits for cross validation.

```r
# Create 10 folds for cross validation
set.seed(200)
my_cv <- vfold_cv(spam_train, v = 10)
```

Since we identified pairs of predictors with opposite correlation to `spam` in the data exploration above, my intuition is to compute a KNN model with these 4 predictors, since the observations near each other might have the same classification of emails. Let's use cross validation to choose the optimal value of k.

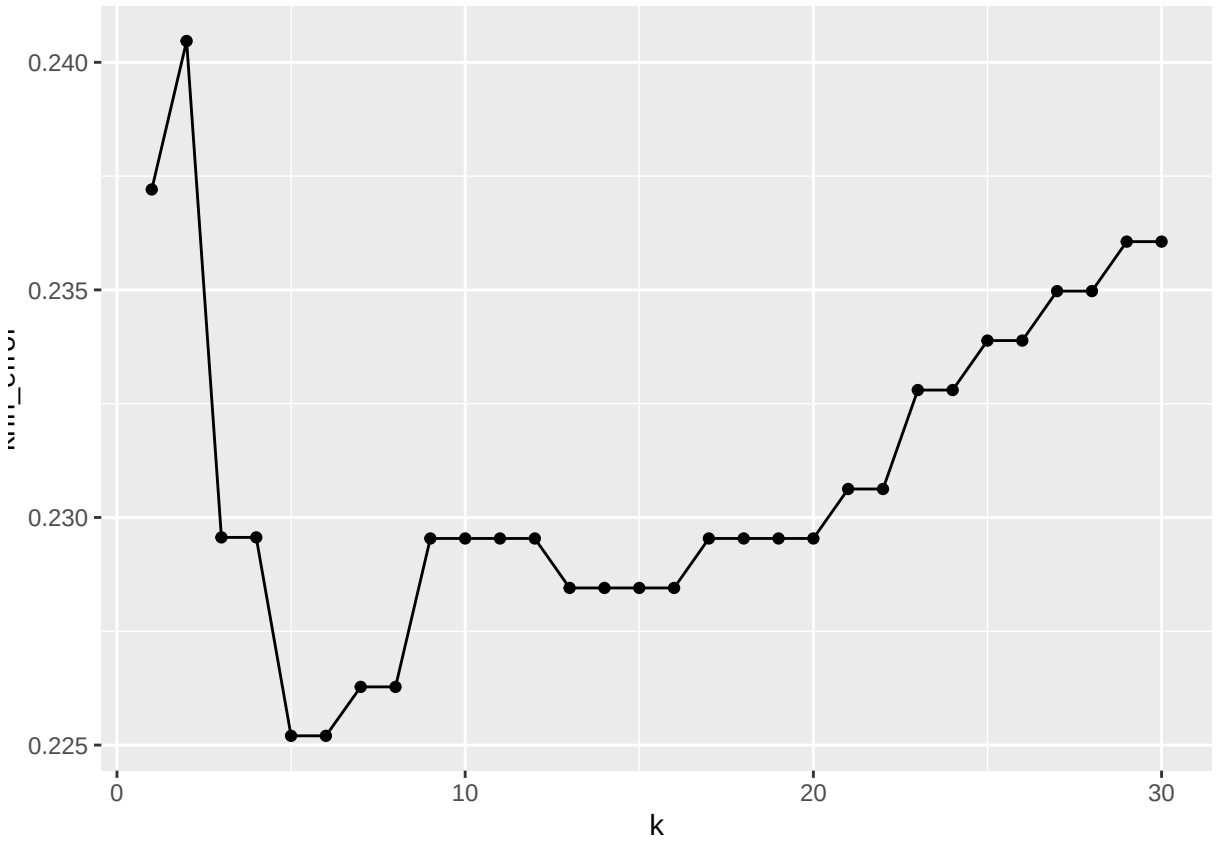
```r
# Fit KNN on the training data to predict spam

spam_knn <- function(k, split){
  model <- kknn(spam ~ word_freq_hp.1 + word_freq_hp + word_freq_receive + word_freq_business,
                train = analysis(split),
                test = assessment(split),
                distance = 1,
                k = k,
                kernel = "rectangular")
  test = assessment(split)
  my_preds <- model$fitted.values
  my_obs <- test$spam
  accuracy <- sum(my_obs==my_preds)/nrow(test)
  error <- 1 - accuracy
  error
}

# Use cross-validation on the training set to select the optimal value of k
my_error_mat <- as.data.frame(matrix(NA, ncol = 30, nrow = 10))
for (i in 1:30){
my_error_mat[,i] <- map_dbl(my_cv$splits, spam_knn, k = i)
}

# Edit column names
colnames(my_error_mat) <- paste("knn",1:30)

# Create plot
knn_error <- map_dbl(my_error_mat, mean)
data.frame(k=1:30, knn_error) %>% ggplot(aes(x=k, y=knn_error)) + geom_line() + geom_point()
```



k = 5 and k = 6 seems to work best at minimizing error. Since I would want to choose an odd k, I would go with k = 5.

Also, to maximize efficiency, I will perform building models and evaluate models simultaneously. This allows me to go back and forth between experiment with adding/omitting predictors and assessing accuracy, sensitivity, and specificity of the models. I would assess each model by calculate each split's accuracy, sensitivity, and specificity through their confidence matrix.

```
# My first model - using KNN
spam_knn_mod <- function(split){
  model <- kknn(spam ~ word_freq_hp.1 + word_freq_hp + word_freq_receive + word_freq_business,
                train = analysis(split),
                test = assessment(split),
                distance = 1,
                k = 5,
                kernel = "rectangular")
  test = assessment(split)
  my_preds <- model$fitted.values
  my_obs <- factor(test$spam, levels = c("1", "0"))
  my_preds <- factor(my_preds, levels = c("1", "0"))
  results <- data.frame(truth = my_obs, estimate = my_preds)
  conf_matrix <- conf_mat(results, truth = truth, estimate = estimate)
  matrix_counts <- conf_matrix$table
  tp <- matrix_counts[1, 1]
  fp <- matrix_counts[1, 2]
  fn <- matrix_counts[2, 1]
  tn <- matrix_counts[2, 2]
  accuracy <- (tp + tn) / sum(matrix_counts)
```

```
  sensitivity <- tp / (tp + fn)
  specificity <- tn / (tn + fp)
  c(Accuracy = accuracy, Sensitivity = as.numeric(sensitivity), Specificity = as.numeric(specificity))
}

# Use map_dfr to apply the function to each split and get a data frame
my_eval_mat <- map_dfr(my_cv$splits, spam_knn_mod)

# Calculate the mean for each column
mean_metrics <- colMeans(my_eval_mat)
mean_metrics
```

```
##    Accuracy Sensitivity Specificity
##   0.7747969   0.5122565   0.9454185
```

The model seems to do pretty good with specificity, but not quite well with accuracy and sensitivity.

After try adding some predictors, upon realizing that all predictors have quite a weak correlation with our response **spam**, we would need the information from many predictors to make valuable predictions. Thus, I would go with backward selection, first starting with all variables and then gradually omit those with weakest correlation while computing their accuracy, sensitivity, and specificity over cross validation.

Since our response is a binary variable, I would want to try applying logistic regression.

```
# My second model - using logistic regression

spam_logistic_mod <- function(split) {
  model <- glm(spam ~ .-spam_num -spam,
               data = analysis(split), family = "binomial")
  test <- assessment(split)
  my_probs <- predict(model, newdata = test, type = "response")
  my_preds <- ifelse(my_probs > 0.5, "1", "0")
  my_obs <- factor(test$spam, levels = c("1", "0"))
  my_preds <- factor(my_preds, levels = c("1", "0"))
  results <- data.frame(truth = my_obs, estimate = my_preds)
  conf_matrix <- conf_mat(results, truth = truth, estimate = estimate)
  matrix_counts <- conf_matrix$table
  tp <- matrix_counts[1, 1]
  fp <- matrix_counts[1, 2]
  fn <- matrix_counts[2, 1]
  tn <- matrix_counts[2, 2]
  accuracy <- (tp + tn) / sum(matrix_counts)
  sensitivity <- tp / (tp + fn)
  specificity <- tn / (tn + fp)
  c(Accuracy = accuracy, Sensitivity = as.numeric(sensitivity), Specificity = as.numeric(specificity))
}

# Use map_dfr to apply the function to each split and get a data frame
my_error_mat <- map_dfr(my_cv$splits, spam_logistic_mod)
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
# Calculate the mean for each column
mean_metrics <- colMeans(my_error_mat)
mean_metrics
```

```
##    Accuracy Sensitivity Specificity
##   0.9151457   0.8757742   0.9400285
```

The logistic regression model seems very promising to me, performing quite well on all three assessments. To achieve our two goals of highest sensitivity and specificity, I would adjusting the threshold accordingly. To increase specificity, we would raise our threshold to 0.9. To increase sensitivity, we would lower our threshold to 0.1.

```r
# SPECIFICITY MODEL
# My third model - using logistic regression

spam_logistic_mod <- function(split) {
  model <- glm(spam ~ .-spam_num -spam,
               data = analysis(split), family = "binomial")
  test <- assessment(split)
  my_probs <- predict(model, newdata = test, type = "response")
  my_preds <- ifelse(my_probs > 0.9, "1", "0")
  my_obs <- factor(test$spam, levels = c("1", "0"))
  my_preds <- factor(my_preds, levels = c("1", "0"))
  results <- data.frame(truth = my_obs, estimate = my_preds)
  conf_matrix <- conf_mat(results, truth = truth, estimate = estimate)
  matrix_counts <- conf_matrix$table
  tp <- matrix_counts[1, 1]
  fp <- matrix_counts[1, 2]
  fn <- matrix_counts[2, 1]
  tn <- matrix_counts[2, 2]
  accuracy <- (tp + tn) / sum(matrix_counts)
  sensitivity <- tp / (tp + fn)
  specificity <- tn / (tn + fp)
  c(Accuracy = accuracy, Sensitivity = as.numeric(sensitivity), Specificity = as.numeric(specificity))
}

# Use map_dfr to apply the function to each split and get a data frame
my_error_mat <- map_dfr(my_cv$splits, spam_logistic_mod)
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
# Calculate the mean for each column
mean_metrics <- colMeans(my_error_mat)
mean_metrics
```

```
##    Accuracy Sensitivity Specificity
##   0.8617893   0.7096642   0.9600884
```

```
# SENSITIVITY MODEL
# My fourth model - using logistic regression

spam_logistic_mod <- function(split) {
  model <- glm(spam ~ .-spam_num -spam,
               data = analysis(split), family = "binomial")
  test <- assessment(split)
  my_probs <- predict(model, newdata = test, type = "response")
  my_preds <- ifelse(my_probs > 0.1, "1", "0")
  my_obs <- factor(test$spam, levels = c("1", "0"))
  my_preds <- factor(my_preds, levels = c("1", "0"))
  results <- data.frame(truth = my_obs, estimate = my_preds)
  conf_matrix <- conf_mat(results, truth = truth, estimate = estimate)
  matrix_counts <- conf_matrix$table
  tp <- matrix_counts[1, 1]
  fp <- matrix_counts[1, 2]
  fn <- matrix_counts[2, 1]
  tn <- matrix_counts[2, 2]
  accuracy <- (tp + tn) / sum(matrix_counts)
  sensitivity <- tp / (tp + fn)
  specificity <- tn / (tn + fp)
  c(Accuracy = accuracy, Sensitivity = as.numeric(sensitivity), Specificity = as.numeric(specificity))
}

# Use map_dfr to apply the function to each split and get a data frame
my_error_mat <- map_dfr(my_cv$splits, spam_logistic_mod)
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
# Calculate the mean for each column
mean_metrics <- colMeans(my_error_mat)
mean_metrics
```

```
##    Accuracy Sensitivity Specificity
##   0.8661968   0.9510968   0.8113608
```

Choosing all predictors seems to work well with logistic regression. I am curious about how this selection performs with naive bayes and its assumption that predictors are not correlated to each other. My assumption is that it will perform relatively poor, given the correlation of our predictors. Let's test that assumption.

```r
# My fifth model - using naive bayes

spam_naive_mod <- function(split) {
  model <- naiveBayes(spam ~ .-spam_num -spam,
              data = analysis(split))
  test <- assessment(split)
  my_preds <- predict(model, test)
  my_probs <- predict(model, test, type = "raw")
  my_obs <- factor(test$spam, levels = c("1", "0"))
  my_preds <- factor(my_preds, levels = c("1", "0"))
  results <- data.frame(truth = my_obs, estimate = my_preds)
  conf_matrix <- conf_mat(results, truth = truth, estimate = estimate)
  matrix_counts <- conf_matrix$table
  tp <- matrix_counts[1, 1]
  fp <- matrix_counts[1, 2]
  fn <- matrix_counts[2, 1]
  tn <- matrix_counts[2, 2]
  accuracy <- (tp + tn) / sum(matrix_counts)
  sensitivity <- tp / (tp + fn)
  specificity <- tn / (tn + fp)
  c(Accuracy = accuracy, Sensitivity = as.numeric(sensitivity), Specificity = as.numeric(specificity))
}

# Use map_dfr to apply the function to each split and get a data frame
my_error_mat <- map_dfr(my_cv$splits, spam_naive_mod)

# Calculate the mean for each column
mean_metrics <- colMeans(my_error_mat)
mean_metrics
```

```
##    Accuracy Sensitivity Specificity
##   0.7030459   0.9348988   0.5520075
```

As expected, naive bayes does not perform well with this selection of predictors, though sensitivity is surprisingly good. Thus, we would want to reduce the number of predictors. Let's try using our pairs of predictors above.

```r
# My sixth model - using naive bayes

spam_naive_mod <- function(split) {
  model <- naiveBayes(spam ~ word_freq_hp.1 + word_freq_hp + word_freq_receive + word_freq_business,
              data = analysis(split))
  test <- assessment(split)
  my_preds <- predict(model, test)
  my_probs <- predict(model, test, type = "raw")
  my_obs <- factor(test$spam, levels = c("1", "0"))
  my_preds <- factor(my_preds, levels = c("1", "0"))
  results <- data.frame(truth = my_obs, estimate = my_preds)
  conf_matrix <- conf_mat(results, truth = truth, estimate = estimate)
  matrix_counts <- conf_matrix$table
  tp <- matrix_counts[1, 1]
  fp <- matrix_counts[1, 2]
  fn <- matrix_counts[2, 1]
```

```
  tn <- matrix_counts[2, 2]
  accuracy <- (tp + tn) / sum(matrix_counts)
  sensitivity <- tp / (tp + fn)
  specificity <- tn / (tn + fp)
  c(Accuracy = accuracy, Sensitivity = as.numeric(sensitivity), Specificity = as.numeric(specificity))
}

# Use map_dfr to apply the function to each split and get a data frame
my_error_mat <- map_dfr(my_cv$splits, spam_naive_mod)

# Calculate the mean for each column
mean_metrics <- colMeans(my_error_mat)
mean_metrics
```

```
##    Accuracy Sensitivity Specificity
##   0.6061873   0.9885779   0.3591651
```

Sensitivity is extremely high in this case, but our accuracy is not maintained. So we would not go with this model.

Since I want to improve my accuracy, I would test with random forest model. After experimenting with backward selection and dropping variables with weak correlation, along with modifying the number of trees and the number of predictors for each tree, I find a model that achieves the dual goal of highest specificity and highest accuracy.

```
# My seventh model - using random forest

set.seed(12)
library(randomForest)

spam_rforest_mod <- function(split) {
  model <- randomForest(spam ~ . -spam_num -word_freq_will -word_freq_parts -word_freq_address -word_fr
                data = analysis(split),
                ntree = 23, mtry = 10)
  test <- assessment(split)
  my_preds <- predict(model, test, type = "class")
  my_obs <- factor(test$spam, levels = c("1", "0"))
  my_preds <- factor(my_preds, levels = c("1", "0"))
  results <- data.frame(truth = my_obs,
                        estimate = my_preds)
  conf_matrix <- conf_mat(results, truth = truth, estimate = estimate)
  matrix_counts <- conf_matrix$table
  tp <- matrix_counts[1, 1]
  fp <- matrix_counts[1, 2]
  fn <- matrix_counts[2, 1]
  tn <- matrix_counts[2, 2]
  accuracy <- (tp + tn) / sum(matrix_counts)
  sensitivity <- tp / (tp + fn)
  specificity <- tn / (tn + fp)
  c(Accuracy = accuracy, Sensitivity = as.numeric(sensitivity), Specificity = as.numeric(specificity))
}

# Use map_dfr to apply the function to each split and get a data frame
```

```r
my_error_mat <- map_dfr(my_cv$splits, spam_rforest_mod)

# Calculate the mean for each column
mean_metrics <- colMeans(my_error_mat)
mean_metrics
```

```
##    Accuracy Sensitivity Specificity
##   0.9412327   0.8983408   0.9676134
```

Model Selection

In this section, evaluate model performance using a variety of metrics. Which models seemed to perform better or worse? Why? Here are some suggestions:

- Use cross-validation as well as training + test sets to evaluate performance
- After assessing performance, revisit models and make small changes
- Consider the structure of the predictors. What relationships do these predictors have? What types of models will tend to work best for these relationships?
- Consider modifications that can be made to increase either sensitivity or specificity.

I performed model building and model selection simultaneously as mentioned above.

Your model

Goals: Identify the three models you feel will best satisfy the following goals:

1. The model with the highest overall accuracy (at least 90%)
2. The model with the highest specificity while maintaining reasonable accuracy (specificity at least 95%, accuracy at least 80%)
3. The model with the highest sensitivity while maintaining reasonable accuracy (sensitivity at least 95%, accuracy at least 80%)

Load the evaluation data using the following code:

```r
spam_eval <-read.csv("spam_eval.csv")
```

Use your 3 models to make 3 sets of predictions on `spam_eval`:

1. Make predictions using the model that best achieves goal 1. Save your predictions as the data frame called `FirstName_LastName_goal1`

2. Make predictions using the model that best achieves goal 2. Save your predictions as the data frame called `FirstName_LastName_goal2`

3. Make predictions using the model that best achieves goal 3. Save your predictions as the data frame called `FirstName_LastName_goal3`

Be sure to replace ***`FirstName_LastName`*** *with your actual first and last names in the above data frame names.*

---

1. Model with best accuracy:

```
best_acc_model <- randomForest(spam ~. -word_freq_will -word_freq_parts -word_freq_address -word_freq_3
                  data = spam_train[,-59],
                  ntree = 23, mtry = 10)

# Make predictions on spam_eval data set
Linh_Vu_goal1 <- predict(best_acc_model, spam_eval)
```

2. Model with best sensitivity:

```
best_sense_model <- glm(spam ~ .,
                        data = spam_train[,-59], family = "binomial")
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
# Make predictions on spam_eval data set
Linh_Vu_goal2_probs <- predict(best_sense_model, spam_eval, type = "response")
Linh_Vu_goal2 <- as.factor(ifelse(Linh_Vu_goal2_probs > 0.1, "1", "0"))
```

3. Model with best specificity:

```
best_spec_model <- randomForest(spam ~. -word_freq_will -word_freq_parts -word_freq_address -word_freq_3
                  data = spam_train[,-59],
                  ntree = 23, mtry = 10)

# Make predictions on spam_eval data set
Linh_Vu_goal3 <- predict(best_spec_model, spam_eval)
```

Once you have made predictions, remove the # symbol from the following lines of code, and replace `FirstName_LastName` with your actual first and last name. **Do not delete the letters .csv at the end of the file name inside the quotations.** Then run the code. This will save your data frames as .csv files in your repo.

```
#write.csv(Linh_Vu_goal1, "Linh_Vu_goal1.csv")
#write.csv(Linh_Vu_goal2, "Linh_Vu_goal2.csv")
#write.csv(Linh_Vu_goal3, "Linh_Vu_goal3.csv")
```

Be sure you commit and push these .csv files to the Github repo in addition to your .Rmd, as they are the predictions I will use to evaluate your model.

Conclusions

Discuss some limitations of your methods and your model. What are some ways you could improve your model if you had more **time**? Identify one variable **not** in the data set you feel could be an important predictor of `spam`. How confident are you in the accuracy of your model?

One big limitation that given more time, I would definitely improve is the clarity of my code. The functions that I used to build models and compute their accuracy, sensitivity, and specificity are quite repetitive and complicated. It would be better to create a function that takes a model and a split as parameters. That way would saves much more lines of code. Also, I would create a heat map if possible to better visualize the variables and their relationships. One variable not in the data set: the number of people recieved the email at the same time

I am about 75% confident of the accuracy of my model.
