Andrea Bruckner
Predict 454 Sec 55

# Individual Assignment #3

## Introduction

In this report I am using the Spambase data set available online at the UC Irvine Machine Learning Repository. The objective for this modeling project is to accurately predict which e-mails are spam and which are not. Although spam can be annoying and at times offensive, a spam filter that accidentally identifies valid e-mails as spam can cause greater issues for a company. After performing a Data Quality Check and Exploratory Data Analysis (EDA), I fit several classification models before selecting four final models. An appendix of R code relevant to the tables and graphics presented is located at the end of this report.

## Data Quality Check

The data used in this report is the Spambase data set available online at the UC Irvine Machine Learning Repository. The data were collected at Hewlett-Packard (HP), and the data set contains 4,601 observations, 57 predictor variables, and one response variable. There are no missing values. **Table 1** provides information about the variables after I performed data conversions. Because the data set contains a lot of variables, **Table 1** shows a summary of the variables rather than a list of each variable.

| Variable | Type | Count | Description |
|---|---|---|---|
| word_freq_WORD | Numeric | 48 | The percentage of words in the e-mail that match WORD. WORD is string of alphanumeric characters (i.e., business, re, 857, etc.). |
| char_freq_CHAR | Numeric | 6 | The percentage of characters in the e-mail that match CHAR. CHAR is a symbol (i.e., !, $, #, etc.). |
| capital_run_length_average | Numeric | 1 | The average length of uninterrupted sequences of capital letters. |
| capital_run_length_longest | Numeric | 1 | The length of longest uninterrupted sequence of capital letters. |
| capital_run_length_total | Numeric | 1 | The total number of capital letters in the e-mail. |
| y | Factor | 1 | The response variable denoting whether the e-mail was considered spam (1) or not spam (0). |

**Table 1: Variables in Spambase Data Set**

With the exception of capital_run_length_longest, capital_run_length_total, and y, all variables were numeric. These three variables were originally integers. I converted the variables capital_run_length_longest and capital_run_length_total to numeric in order to facilitate analysis. I converted y to a factor variable with two levels: Not_Spam and Spam. Given that the purpose of this data set is to determine whether a given e-mail was spam or not, this is clearly a classification problem.

**Figure 1** provides a visual of the number of Spam v. Not_Spam e-mails. In total, 2,788 e-mails were considered Not_Spam while 1,813 e-mails were Spam. Thus, Spam e-mails account for approximately 40 percent of the e-mails collected by HP.

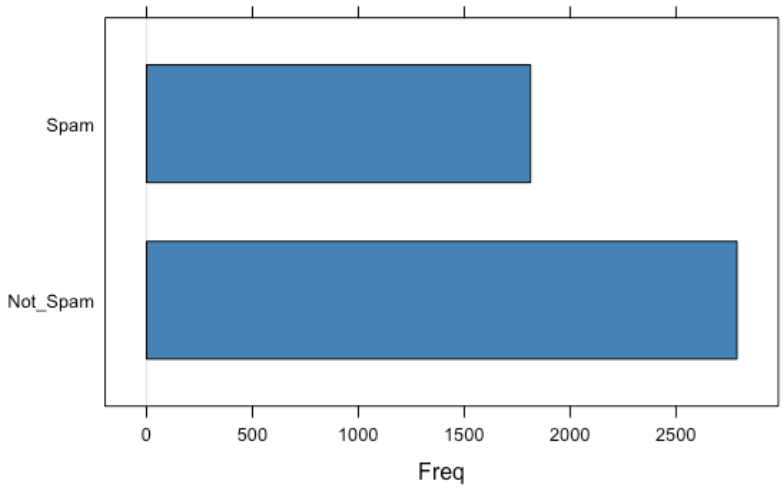


**Figure 1: Frequency of Spam v. Not_Spam**

After converting several variables to other data types, I created histograms to get a better idea of the distributions for each predictor. **Figure 2** shows plots for just a few of the predictor variables. Overall, the histograms for each predictor variable resembled the ones below.

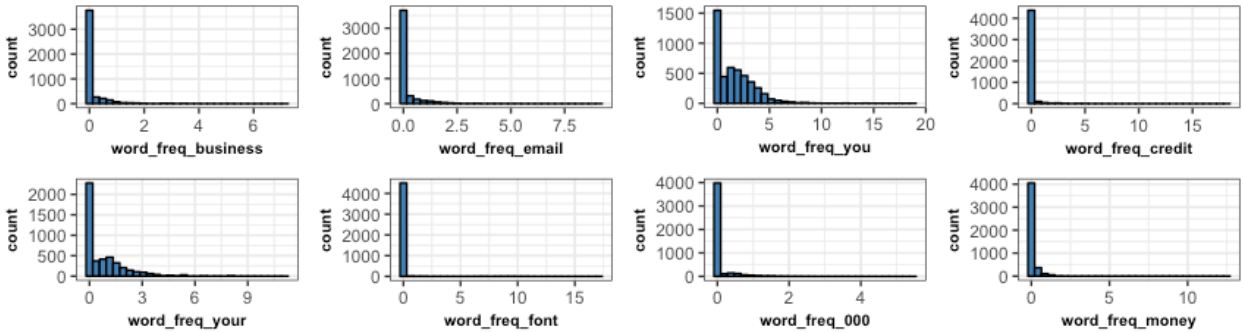


**Figure 2: Distributions for Some Variables**

Because most of the variables represent frequencies of a particular word or character in an e-mail, the majority of observations fall near zero. This makes detecting outliers very difficult because observations containing a high frequency of a specific word or character are not necessarily outliers. Given the type of data set I am working with and the goal for the data, I have decided to not consider any values outliers. I have also noted that all the data seem to be valid since none of the values are negative, which would be impossible given what each variable represents.

## Exploratory Data Analysis

After taking inventory of the data to make sure the data are correct and of good quality, I conducted a more in-depth analysis, known as EDA, to detect interesting relationships in the data. In the previous section, I mentioned that the type of statistical problem presented by the data is a classification problem. I decided to create box plots of each predictor in order to better understand their relationships with the response variable. I found that applying a natural log transformation to the predictors made the plots easier to interpret; otherwise, the box plots mostly resembled lines rather than boxes at the zero mark. **Figure 3** shows several predictors that I thought had interesting relationships with the different classes of the response variable.

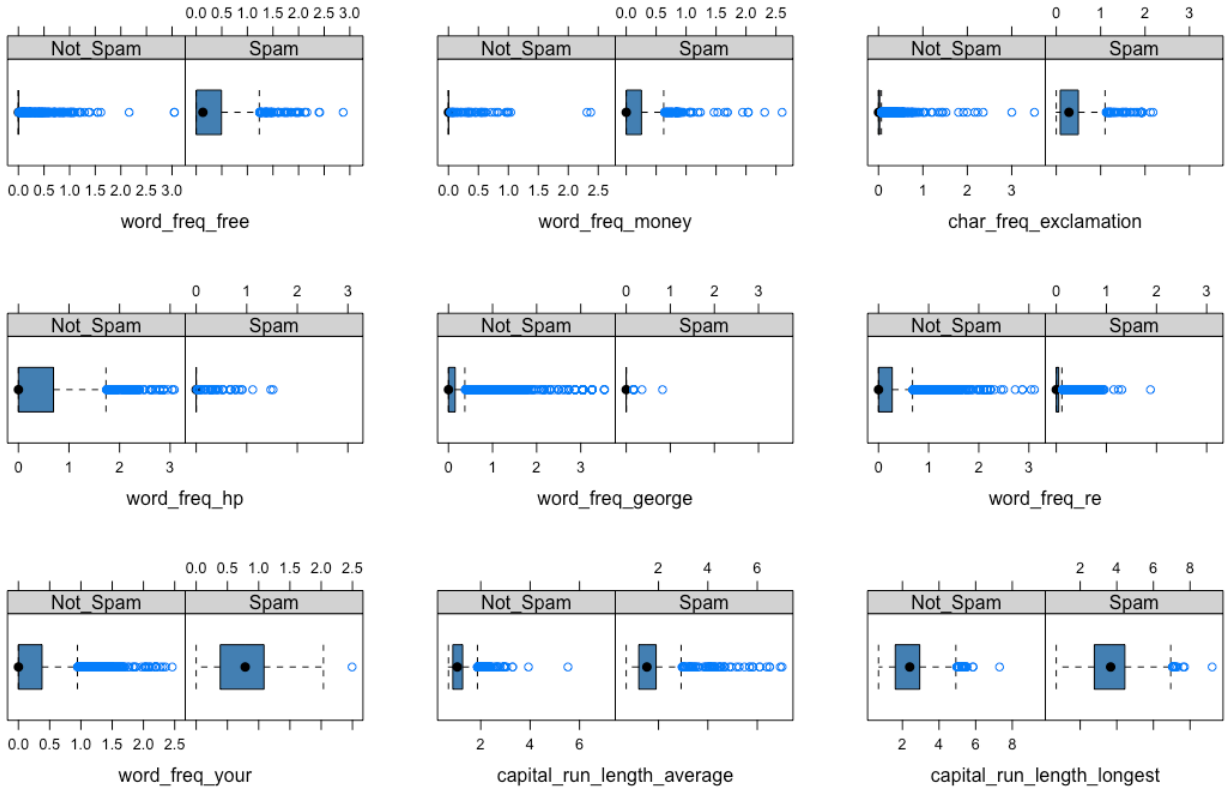


**Figure 3: Box Plots of Select Predictors**

The first row of plots in the figure above depicts words or characters that had a higher occurrence in Spam than Not_Spam. It seems logical that “free,” “money,” and “!” would appear more in spam e-mails than business e-mails. The plots in the second row show the opposite. Given that the data set is based on e-mails from HP, it is reasonable for “hp” to appear more frequently in Not_Spam than Spam. It also seems more likely that Not_Spam e-mail would address a person by name (“George”) or contain “re” since people are replying to each other’s e-mails in the company. As for the last row of plots, these all depict variables that have a distinct relationship with the response classes, even if the relationship is not as stark as in the first couple rows of plots. Each variable in the last row of plots has a higher interquartile range for Spam than Not_Spam, and there are overall more higher values for these variables for Spam than for Not_Spam. The variable plots shown above are just a few of many variables that exhibited interesting relationships with the response variable. Other variables, such as word_freq_you, char_freq_usd, word_freq_remove, etc., resembled the plots above but were excluded since the selected plots conveyed nearly the same information.

In addition to creating plots, I fit a tree model to get a quick overview of potentially important predictors in the data. The tree model in **Figure 4** shows the relationships of several untransformed predictors with the response variable. (I found that the tree for the predictors with a log transformation was nearly identical, so I will not discuss it further.) The tree model contains only six of 57 possible predictors, indicating that the select predictors included in the tree are likely very important in predicting the class of the response variable. With exception of word_freq_hp, e-mails that contained a high frequency of any of the predictors shown in the tree generally were classified as Spam. Several of the predictors in the tree are the same as ones that appear in **Figure 3**. For instance, the tree model reinforces the idea that a high frequency of the word “free” is likely Spam.

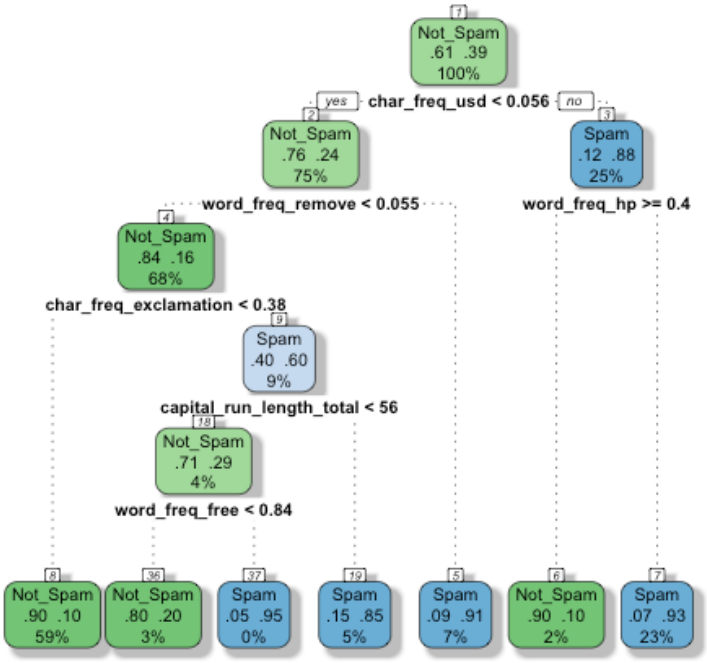


**Figure 4: Tree Model of All Data**

Conducting EDA helped me identify interesting relationships and provided insights that I can use when building models. Given that most of the variables identified in the tree model also showed interesting relationships with the response classes when I plotted them, I expect these variables will appear in the models I create in the next section.

## Models and Analysis

After conducting a Data Quality Check and EDA, I split my data set in order to use cross-validation to build and evaluate models. I used a random number generator to allocate 70 percent of the data to the training set and the remaining 30 percent to the test set. I decided to create two testing and training sets using the same seed so that I could build and compare how models performed using either the untransformed predictor variables or the logarithmically transformed predictor variables. The untransformed and log transformed training sets each contained 3,221 observations, and the test sets each contained 1,380 observations.

When building models, I used functions from a variety of R packages in order to explore their options and performance. I will discuss only the models that performed best.

### Model 1: Logistic Regression Model Using Stepwise Variable Selection

The first model I fit after exploring the data was a logistic regression model. I tried several variable selection algorithms before deciding to use a stepwise selection based on minimizing the AIC value. I discovered that the model performed best on the logarithmically transformed data. **Table 2** below shows the coefficient estimates for the 39 predictors selected for this model. All of the variables featured in the box plots and naïve tree model discussed in the EDA section appear in this model. The variables discussed in the EDA section are also all significant, except for capital_run_length_longest. It would be interesting to see how the model

performance changes if I created another model using only the significant predictors. This is something I could try if I decide to work on the data set again in the future.

| Variable | Estimate | Std. Error | z value | Pr(>\|z\|) | Significance |
|---|---|---|---|---|---|
| (Intercept) | -5.3309 | 0.4110 | -12.971 | < 2e-16 | *** |
| capital_run_length_longest | 0.1329 | 0.1761 | 0.755 | 0.45019900 | |
| word_freq_hp | -5.3451 | 0.9016 | -5.928 | 3.06E-09 | *** |
| word_freq_remove | 2.8306 | 0.4988 | 5.674 | 1.39E-08 | *** |
| char_freq_usd | 5.4794 | 1.0016 | 5.471 | 4.49E-08 | *** |
| word_freq_free | 1.5916 | 0.2562 | 6.213 | 5.2E-10 | *** |
| word_freq_george | -11.0782 | 2.5433 | -4.356 | 0.00001330 | *** |
| word_freq_edu | -2.6399 | 0.5440 | -4.852 | 0.00000122 | *** |
| char_freq_exclamation | 2.2196 | 0.2858 | 7.766 | 8.08E-15 | *** |
| word_freq_our | 1.4757 | 0.2479 | 5.952 | 2.65E-09 | *** |
| word_freq_meeting | -4.2303 | 1.3627 | -3.104 | 0.00190800 | ** |
| word_freq_business | 1.6117 | 0.4627 | 3.483 | 0.00049500 | *** |
| word_freq_000 | 2.0251 | 0.6476 | 3.127 | 0.00176400 | ** |
| word_freq_money | 2.3107 | 0.6664 | 3.467 | 0.00052500 | *** |
| capital_run_length_total | 0.6780 | 0.1186 | 5.715 | 0.00000001 | *** |
| word_freq_conference | -4.5997 | 1.9685 | -2.337 | 0.01945600 | * |
| word_freq_internet | 1.3783 | 0.3815 | 3.613 | 0.00030200 | *** |
| word_freq_cs | -41.1898 | 34.1304 | -1.207 | 0.22749600 | |
| word_freq_re | -1.2928 | 0.3317 | -3.898 | 0.00009710 | *** |
| word_freq_data | -1.8198 | 0.7049 | -2.582 | 0.00983100 | ** |
| char_freq_semicolon | -2.1116 | 0.6332 | -3.335 | 0.00085400 | *** |
| word_freq_credit | 1.9559 | 0.8159 | 2.397 | 0.01651700 | * |
| word_freq_project | -2.5548 | 1.0277 | -2.486 | 0.01291900 | * |
| char_freq_pound | 2.1250 | 1.3606 | 1.562 | 0.11833800 | |
| word_freq_650 | 1.9252 | 0.5299 | 3.633 | 0.00028000 | *** |
| word_freq_85 | -2.7699 | 1.3856 | -1.999 | 0.04560200 | * |
| word_freq_your | 0.5353 | 0.1766 | 3.031 | 0.00244100 | ** |
| word_freq_415 | -15.8209 | 5.2630 | -3.006 | 0.00264700 | ** |
| word_freq_make | -1.0501 | 0.4938 | -2.126 | 0.03347400 | * |
| word_freq_original | -3.6931 | 2.0134 | -1.834 | 0.06661300 | . |
| char_freq_l_paren | -1.3442 | 0.6083 | -2.210 | 0.02712300 | * |
| word_freq_over | 1.2769 | 0.5222 | 2.445 | 0.01448500 | * |
| capital_run_length_average | 0.6121 | 0.3063 | 1.998 | 0.04571100 | * |
| word_freq_technology | 1.1874 | 0.5947 | 1.997 | 0.04585400 | * |
| word_freq_hpl | -1.6194 | 1.0318 | -1.570 | 0.11652500 | |
| word_freq_order | 0.9031 | 0.5967 | 1.513 | 0.13017700 | |
| word_freq_report | 0.7500 | 0.4451 | 1.685 | 0.09197000 | . |
| word_freq_people | -0.7856 | 0.4781 | -1.643 | 0.10036300 | |
| word_freq_lab | -1.9911 | 1.6562 | -1.202 | 0.22928000 | |
| word_freq_will | -0.3046 | 0.2119 | -1.438 | 0.15055100 | |
| Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1 | | | | | |

**Table 2: Coefficient Estimates**

After reviewing the coefficient estimates, I created a ROC curve to assess the fit and predictive ability of the model. **Figure 5** shows the ROC curve based on the training set predictions. The curve hugs the upper left corner of the plot, which indicates that the model has a strong ability to

distinguish between Spam and Not_Spam. The area under the curve (AUC) is 0.9847, which represents the probability of the model correctly predicting the response class when given a random pair of observations representing each class.

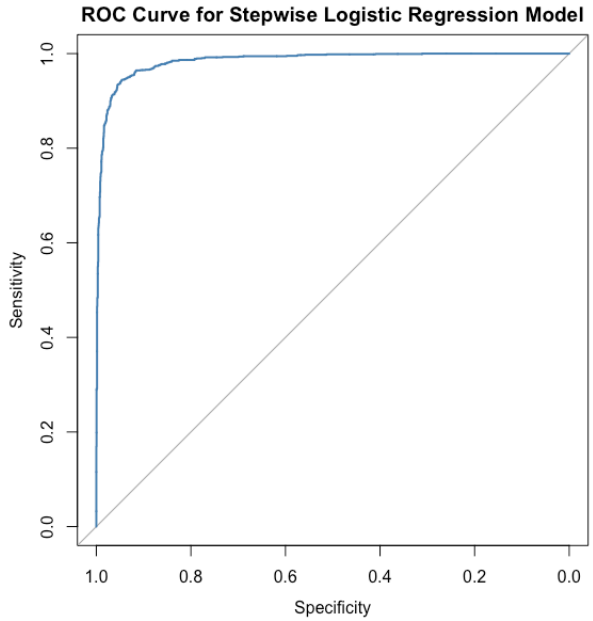


**Figure 5: ROC Curve for Stepwise Logistic Regression Model**

## Model 2: Tree Model

The next model I created was a tree model. I fit a tree model on each the untransformed and transformed training data. Both models performed the same, so I will discuss only one model. **Figure 6** shows a plot of the tree model trained on the untransformed data set. Each node in the tree states three things: the majority class, class proportions, and total percentage of the data set considered for the split. For instance, the top node shows that the majority of the observations are Not_Spam—specifically, 61 percent are Not_Spam while 39 percent are Spam—and that 100 percent of the data set is present in the node. The decision rule at this first node is based on char_freq_usd. We can see that observations with a char_freq_usd greater than 0.044 are likely Spam.

As the tree shows, most of the predictors identified as potentially important in the naïve tree plot in **Figure 4** are present in this plot as well. The only variable that is not part of this tree is capital_run_length_total. Instead, capital_run_length_average appears, along with some other variables that showed predictive promise during the EDA. Both the naïve EDA tree and the tree fit to the training data selected the same top three predictors: char_freq_usd, word_freq_remove, and word_freq_hp.

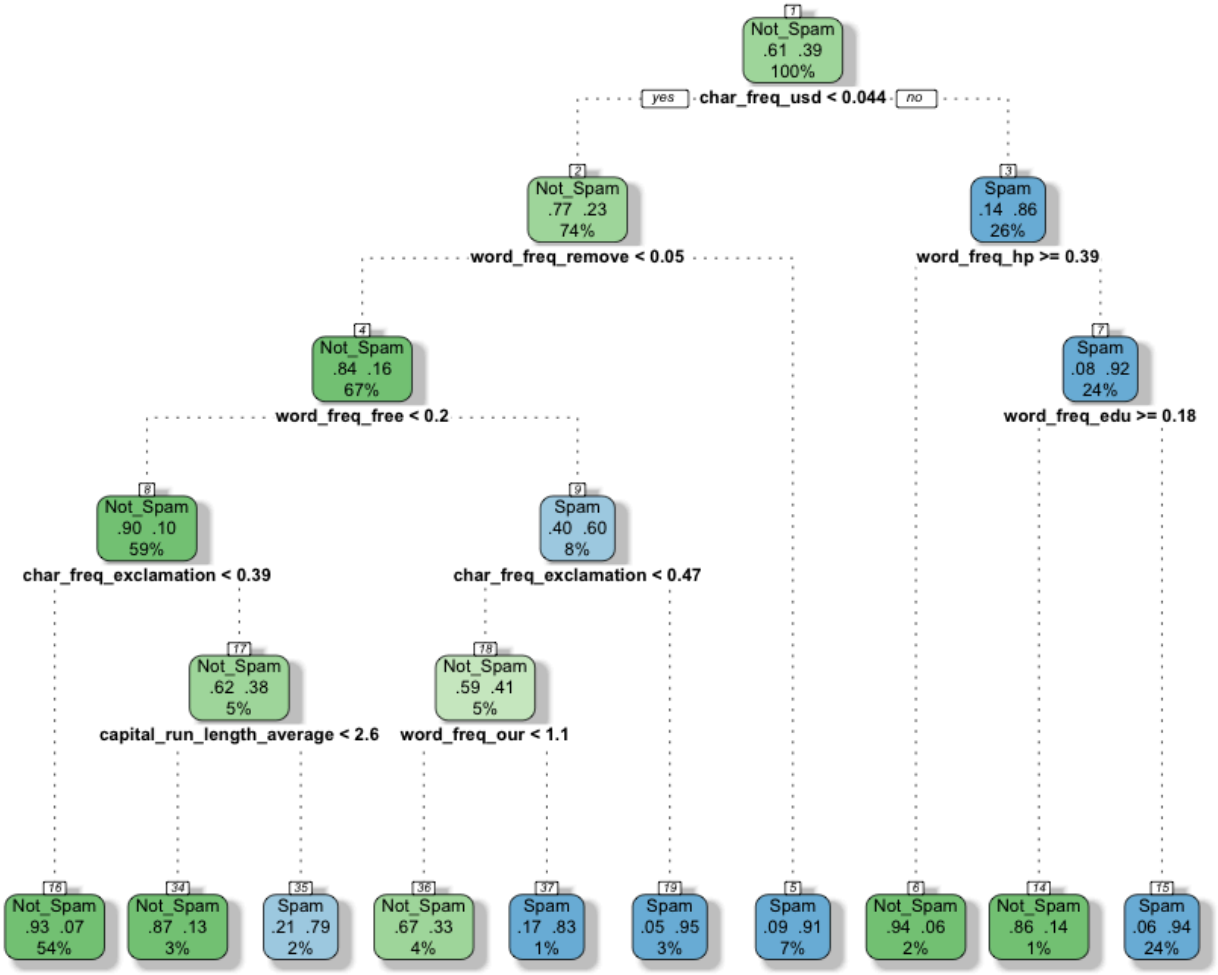


**Figure 6: Tree Plot**

After examining the tree, I decided to create a ROC curve to assess the fit of the model and get an idea of how its performance compares to the stepwise logistic regression model. **Figure 7** shows that the curve is not quite as close to the upper left corner of the plot as the ROC curve for the previous model, suggesting that its ability to distinguish between Spam and Not_Spam is not as strong. The AUC confirms this as it is 0.9223, which is considerably less than the 0.9847 AUC for the stepwise logistic regression model.

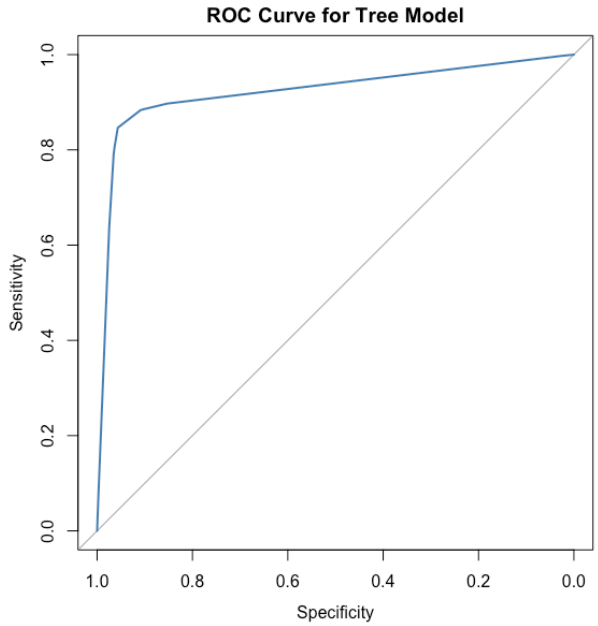


**Figure 7: ROC Curve for Tree Model**

## Model 3: SVM Model

A support vector machine (SVM) is the third model I fit. I used 10-fold cross-validation for this model, as well as a Gaussian Radial Basis kernel. When fit on the untransformed training data, the model had a total of 1,082 support vectors and a training error rate of 0.044396. The model fit on the log transformed training data had 829 support vectors and a training error rate of 0.043775. The model seemed to perform better using the log transformations, so it is the one I used to create a ROC curve. **Figure 8** shows that the curve hugs the upper left corner, suggesting that the model is good at predicting the response variable. The AUC for this model is 0.9879, which is slightly better than the AUC for the stepwise logistic regression model.

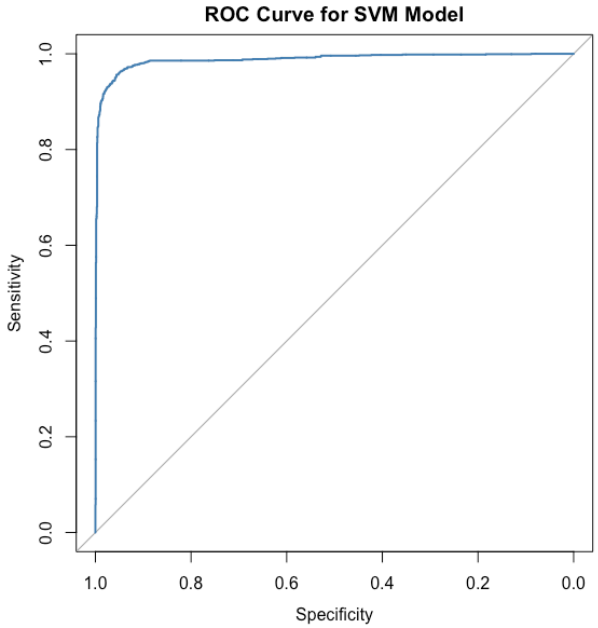


**Figure 8: ROC Curve for SVM Model**

## Model 4: Random Forest Model

The last model I fit was a random forest model. I created a forest of 500 trees using 10-fold cross-validation. Each split tried 29 variables. I found the model performed better on the untransformed training set than the log transformed training set, so this is the model I will discuss. It had a 4.78 percent out-of-bag error rate estimate. In **Figure 9** we can see a plot showing the importance of each variable. The top 14 variables are all ones that have appeared as important at various points in the report, either in the EDA section or in previous models. Just like in the tree plots, char_freq_usd is identified as the most important predictor variable.

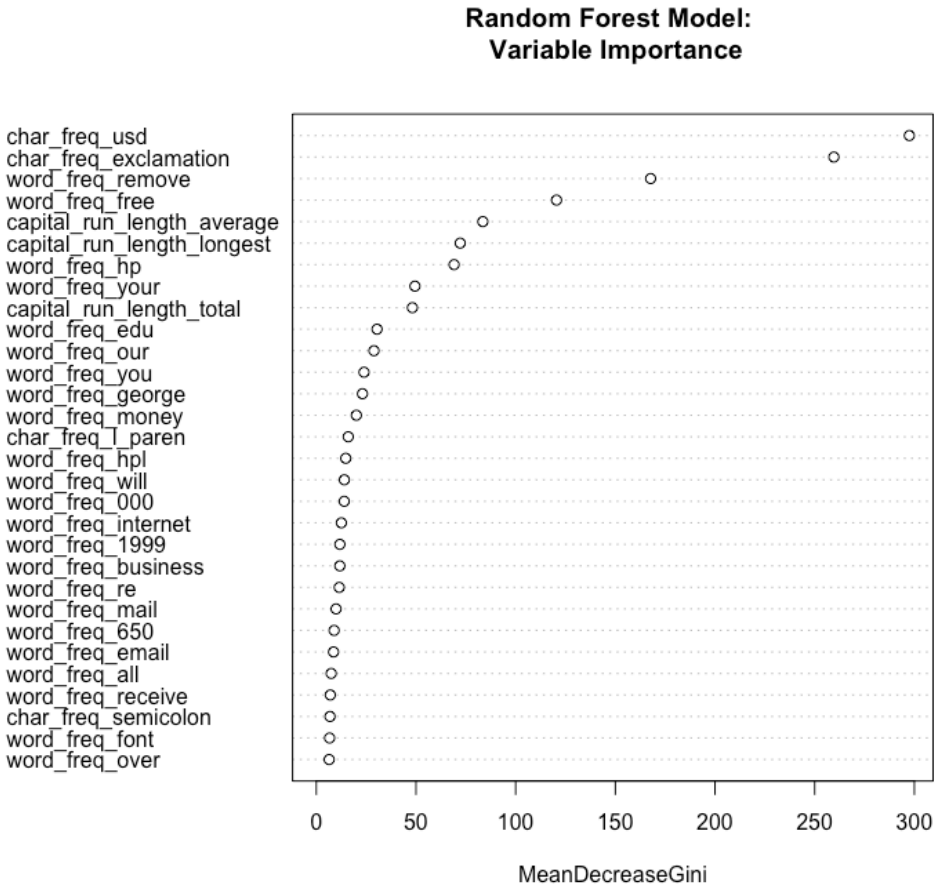


**Figure 9: Variable Importance Plot**

As I have done with the three previous models, I created a ROC curve for the random forest model as well. **Figure 10** shows that the curve seemingly perfectly fits the upper left corner; its AUC value of 1 confirms a perfect fit. While a curve that hugs the upper left corner is ideal, a curve that fits the corner perfectly is suspicious. It suggests I may have overfit the model to the training data. Seeing how the model performs on the test data will determine whether or not this model is actually useful for predicting the response.

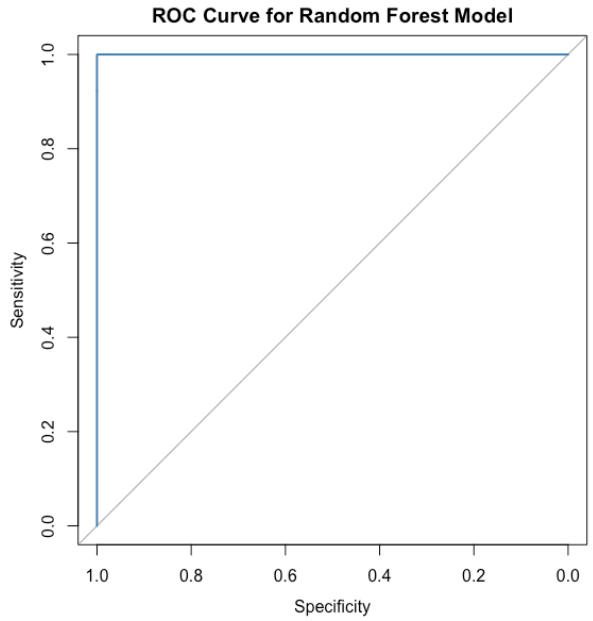


**Figure 10: ROC Curve for Random Forest Model**

## Model Comparison

Once I fit all my models, I created confusion matrices of their predictions. **Table 3** shows the confusion matrices for each model based on both the training and test data. We can see that many models had similar, if not the same, predictions sometimes. The tree model did, however, falsely predict more valid e-mails as Spam than any other model.

| Model | Actual/Predicted | In-Sample Performance | | Out-of-Sample Performance | |
|---|---|---|---|---|---|
| | | Not_Spam | Spam | Not_Spam | Spam |
| Log Stepwise Logistic Regression | Not_Spam | 1873 | 97 | 800 | 57 |
| | Spam | 85 | 1166 | 30 | 493 |
| Tree | Not_Spam | 1873 | 194 | 794 | 101 |
| | Spam | 85 | 1069 | 36 | 449 |
| Log SVM | Not_Spam | 1898 | 83 | 807 | 52 |
| | Spam | 60 | 1180 | 23 | 498 |
| Random Forest | Not_Spam | 1957 | 0 | 807 | 59 |
| | Spam | 1 | 1263 | 23 | 491 |

**Table 3: Confusion Matrices**

After examining the confusion matrices, I calculated the accuracy and false positive rate (FPR) for each model to evaluate their performance both in-sample and out-of-sample. Accuracy was an obvious metric to use, but I decided to use FPR too because e-mail falsely labeled as spam has greater potential to cause issues for a company than occasional spam accidentally passing through the spam filter. Businesses do not want important e-mails to get filtered out and never make it to the recipient. **Table 4** provides a summary of accuracy and FPR for each model.

| Model | In-Sample Performance | | Out-of-Sample Performance | |
|---|---|---|---|---|
| | Accuracy | False Positive Rate | Accuracy | False Positive Rate |
| Log Stepwise Logistic Regression | 0.9435 | 0.049238579 | 0.9370 | 0.066511085 |
| Tree | 0.9134 | 0.093855830 | 0.9007 | 0.112849162 |
| Log SVM | 0.9556 | 0.041898031 | 0.9457 | 0.060535506 |
| Random Forest | 0.9997 | 0 | 0.9406 | 0.068129330 |

**Table 4: Summary of Model Performance**

Although the random forest model seemed best based on its in-sample performance, it was indeed overfit to the training data and performed worse on the test data. In terms of accuracy, the SVM model using log transformed predictors performed best on the test data, and second best on the training data. The stepwise logistic regression model using log transformed data performed third best both in-sample and out-of-sample in terms of accuracy, and the tree model had the worst accuracy of all the models for both training and test data. For the most part, the models' FPR performance aligns with their performance in terms of accuracy; however, it is interesting to note the out-of-sample FPR for each model. The SVM model performed best, but the stepwise logistic regression model performed second best. This shows that accuracy and FPR are not always in tandem. If a company is concerned with valid e-mails getting stuck in the spam filter, the company is better off using the logistic regression model than the random forest model. However, the best model overall in terms of out-of-sample performance is the SVM model. It has the best accuracy and FPR of all the models for the test data.

## Conclusion

This report demonstrated various classification modeling techniques for predicting whether or not an e-mail was spam. After assessing the quality of the data, I explored interesting relationships in the data by creating plots and fitting a naïve tree model. I fit various models using both untransformed and log transformed predictor variables. I then fit four final models: a logistic regression model using stepwise variable selection, a tree model, an SVM model, and a random forest model. The SVM model using log transformed predictors performed the best.

## Appendix A: Relevant R Code

```
############################################################
# Load Data
############################################################

# Read data
data <- read.csv("https://archive.ics.uci.edu/ml/machine-learning-
databases/spambase/spambase.data", header=F)

############################################################
# Data Quality Check
############################################################

# Explore the data
dim(data) # 4601   58
names(data) # need to be renamed
str(data) # all numeric except last 3 are integer
summary(data) # no missing data, no invalid data (negative values, in this case)

# List of all column names
all.col.names <- c("word_freq_make",
                   "word_freq_address",
                   "word_freq_all",
                   "word_freq_3d",
                   "word_freq_our",
                   "word_freq_over",
                   "word_freq_remove",
                   "word_freq_internet",
                   "word_freq_order",
                   "word_freq_mail",
                   "word_freq_receive",
                   "word_freq_will",
                   "word_freq_people",
                   "word_freq_report",
                   "word_freq_addresses",
                   "word_freq_free",
                   "word_freq_business",
                   "word_freq_email",
                   "word_freq_you",
                   "word_freq_credit",
                   "word_freq_your",
                   "word_freq_font",
                   "word_freq_000",
                   "word_freq_money",
                   "word_freq_hp",
                   "word_freq_hpl",
                   "word_freq_george",
                   "word_freq_650",
                   "word_freq_lab",
```

```
                "word_freq_labs",
                "word_freq_telnet",
                "word_freq_857",
                "word_freq_data",
                "word_freq_415",
                "word_freq_85",
                "word_freq_technology",
                "word_freq_1999",
                "word_freq_parts",
                "word_freq_pm",
                "word_freq_direct",
                "word_freq_cs",
                "word_freq_meeting",
                "word_freq_original",
                "word_freq_project",
                "word_freq_re",
                "word_freq_edu",
                "word_freq_table",
                "word_freq_conference",
                "char_freq_semicolon",
                "char_freq_l_paren",
                "char_freq_l_bracket",
                "char_freq_exclamation",
                "char_freq_usd",
                "char_freq_pound",
                "capital_run_length_average",
                "capital_run_length_longest",
                "capital_run_length_total",
                "y")

# Rename all columns
colnames(data) <- all.col.names
names(data)

# Convert to numeric
data$capital_run_length_longest <- as.numeric(data$capital_run_length_longest)
data$capital_run_length_total  <- as.numeric(data$capital_run_length_total )

# Convert to factor
data$y <- as.factor(data$y)
levels(data$y) <- c("Not_Spam", "Spam")
barchart(data$y, col = "steelblue")

# Create log transformations of all predictors
pred.log <- lapply(data[1:57], log1p) # log1p works better than log when x is small
pred.log <- data.frame(pred.log, y = data$y)
head(pred.log)

# Plot histograms of the variables--See how to do this in lattice after PlotsD
plot_vars <- function (data, column){
  ggplot(data = data, aes_string(x = column)) +
```

```
   geom_histogram(color =I("black"), fill = I("steelblue"))+
   xlab(column) + theme_bw() + theme(axis.title=element_text(size=8, face="bold"))
}

# Histograms for report—took snip of just some for report
plotsA <- lapply(colnames(data[1:24]), plot_vars, data = data)
length(plotsA)
do.call("grid.arrange", c(plotsA, ncol=4))

#########################################################
# EDA
#########################################################

# Create box plots of each predictor according to response
myBoxplots <- function(variable, df){
  toPlot = paste0("~ ", variable, " | y")
  bwplot(as.formula(toPlot), data = df,
         layout = c(2, 1),
         par.settings = list(
           box.umbrella=list(col= "black"),
           box.dot=list(col= "black"),
           box.rectangle = list(col= "black", fill = "steelblue")),
         strip = strip.custom(bg="lightgrey"),
         xlab = paste0(variable))}

# Interesting boxplots for discussion
plots.report <- lapply(colnames(pred.log[c(16,24,52,25,27,45,21,55,56)]), myBoxplots, pred.log)
length(plots.report)
do.call("grid.arrange", c(plots.report, ncol=3))

# Create naive tree models
set.seed(123)
fancyRpartPlot(rpart(y ~ ., data = data), sub = "")

#########################################################
# Model Build - Set Up
#########################################################

# Create train/test sets (70/30)
nrow(data) # 4601: 0.70*4601 = 3220.7 (train should have 3220 or 3221 observations)

set.seed(123)
train <- sample_frac(data, 0.70)
train_id <- as.numeric(rownames(train))
test <- data[-train_id,]

dim(train) # 3221   58
dim(test) # 1380   58

# Create log train/test sets (70/30)
set.seed(123)
```

Andrea Bruckner
Predict 454 Sec 55

```
train.log <- sample_frac(pred.log, 0.70)
train_id.log <- as.numeric(rownames(train.log))
test.log <- pred.log[-train_id.log,]

dim(train.log) # 3221   58
dim(test.log) # 1380   58

#########################################################
# Model Build -- Fit Model Suite
#########################################################

# (1) a logistic regression model using variable selection
ptm <- proc.time() # Start the clock!
logit.control.log <- trainControl(classProbs = T, savePred = T , verboseIter = T)
set.seed(123)
model.logit.step2.log <- train(y ~ ., data = train.log, method = "glmStepAIC",
                      direction = "forward", trControl = logit.control.log)
proc.time() - ptm # Stop the clock
summary(model.logit.step2.log$finalModel)

# Predict train
set.seed(123)
model.logit.step2.pred.log <- predict(model.logit.step2.log, newdata = train.log,
                              type = "prob")[,2]

# Plot ROC curve
set.seed(123)
model.logit.step2.roc.log <- plot.roc(train.log$y, model.logit.step2.pred.log)
model.logit.step2.auc.log <- model.logit.step2.roc.log$auc
model.logit.step2.auc.log # Area under the curve: 0.9847

par(pty = "s") # "s" generates a square plotting region
plot(model.logit.step2.roc.log, col = "steelblue", main = "ROC Curve for Stepwise Logistic
Regression Model")
par(pty = "m") # "m" generates the maximal plotting region

# Predict train for confusion matrix
set.seed(123)
model.logit.step2.pred2.log <- predict(model.logit.step2.log, newdata = train.log) # no type =
"prob"
set.seed(123)
model.logit.step2.cmat.log <- confusionMatrix(model.logit.step2.pred2.log, train.log$y)
model.logit.step2.cmat.log

# Predict test
set.seed(123)
model.logit.step2.pred.test.log <- predict(model.logit.step2.log, newdata = test.log)

set.seed(123)
model.logit.step2.cmat.test.log <- confusionMatrix(model.logit.step2.pred.test.log, test.log$y)
model.logit.step2.cmat.test.log
```

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

# (2) a tree model -- used model.tree in report -- does way better than model.tree2

ptm <- proc.time() # Start the clock!
set.seed(123)
model.tree <- rpart(y ~ ., data = train)
model.tree
proc.time() - ptm # Stop the clock

dev.new(width=10, height=8) # This fixes the tiny font issue
model.tree_plot <- fancyRpartPlot(model.tree, sub = "") # The font is very small without the
above code
dev.off()

summary(model.tree)

# Predict train
set.seed(123)
model.tree.pred <- predict(model.tree, newdata = train,
                           type = "prob")[,2]

# Plot ROC curve
set.seed(123)
model.tree.roc <- plot.roc(train$y, model.tree.pred)
model.tree.auc <- model.tree.roc$auc
model.tree.auc # Area under the curve: 0.9223

par(pty = "s") # "s" generates a square plotting region
plot(model.tree.roc, col = "steelblue", main = "ROC Curve for Tree Model")
par(pty = "m") # "m" generates the maximal plotting region

# Predict train for confusion matrix
set.seed(123)
model.tree.pred2 <- predict(model.tree, newdata = train)

head(model.tree.pred2) # This needs to show just Spam or Not_Spam--shows matrix with
probabilities of each prediction
#######
# Make predictions just Spam or Not_Spam
model.tree.pred2.binary <- data.frame(model.tree.pred2)

model.tree.pred2.binary$y <- NA

# Convert to factor
model.tree.pred2.binary$y <- as.factor(model.tree.pred2.binary$y)
levels(model.tree.pred2.binary$y) <- c("Not_Spam", "Spam")

for (i in 1:nrow(model.tree.pred2.binary)){
```

```
  model.tree.pred2.binary[i,3] <- ifelse(model.tree.pred2.binary[i,1] >
model.tree.pred2.binary[i,2], "Not_Spam", "Spam")
}

#######
set.seed(123)
model.tree.cmat <- confusionMatrix(model.tree.pred2.binary$y, train$y)
model.tree.cmat

# Predict test
set.seed(123)
model.tree.pred.test <- predict(model.tree, newdata = test)
head(model.tree.pred.test) # This needs to show just Spam or Not_Spam
#######
# Make predictions just Spam or Not_Spam
model.tree.pred.test.binary <- data.frame(model.tree.pred.test)

model.tree.pred.test.binary$y <- NA

# Convert to factor
model.tree.pred.test.binary$y <- as.factor(model.tree.pred.test.binary$y)
levels(model.tree.pred.test.binary$y) <- c("Not_Spam", "Spam")

for (i in 1:nrow(model.tree.pred.test.binary)){
  model.tree.pred.test.binary[i,3] <- ifelse(model.tree.pred.test.binary[i,1] >
model.tree.pred.test.binary[i,2], "Not_Spam", "Spam")
}

set.seed(123)
model.tree.cmat.test <- confusionMatrix(model.tree.pred.test.binary$y, test$y)
model.tree.cmat.test

# ------------------------------------------------------------------------#

# (3) a Support Vector Machine

ptm <- proc.time() # Start the clock!
svm.control.log <- trainControl(method = "cv", classProbs = T, savePred = T, verboseIter = T)
set.seed(123)
model.svm.CV.log <- train(y ~ ., data = train.log, method = "svmRadial",
                  trControl = svm.control.log) # metric="ROC" doesn't do anything for this model
proc.time() - ptm # Stop the clock

model.svm.CV.log$finalModel

# Predict train
set.seed(123)
model.svm.CV.pred.log <- predict(model.svm.CV.log, newdata = train.log,
                        type = "prob")[,2]

# Plot ROC curve
```

```
set.seed(123)
model.svm.CV.roc.log <- plot.roc(train.log$y, model.svm.CV.pred.log)
model.svm.CV.auc.log <- model.svm.CV.roc.log$auc
model.svm.CV.auc.log # Area under the curve: 0.9879

par(pty = "s") # "s" generates a square plotting region
plot(model.svm.CV.roc.log, col = "steelblue", main = "ROC Curve for SVM Model")
par(pty = "m") # "m" generates the maximal plotting region

# Predict train for confusion matrix
set.seed(123)
model.svm.CV.pred2.log <- predict(model.svm.CV.log, newdata = train.log) # no type = "prob"
set.seed(123)
model.svm.CV.cmat.log <- confusionMatrix(model.svm.CV.pred2.log, train.log$y)
model.svm.CV.cmat.log

# Predict test
set.seed(123)
model.svm.CV.pred.test.log <- predict(model.svm.CV.log, newdata = test.log)

set.seed(123)
model.svm.CV.cmat.test.log <- confusionMatrix(model.svm.CV.pred.test.log, test.log$y)
model.svm.CV.cmat.test.log

# ------------------------------------------------------------------------#

# (4) Random Forest

# Create RF model
set.seed(123)
control.rf2 <- trainControl(method = "cv", classProbs = T, savePred = T, verboseIter = T)
ptm <- proc.time() # Start the clock!
set.seed(123)
rf_random2 <- train(y ~ ., data = train, method="rf", metric="Accuracy", trControl=control.rf2)
proc.time() - ptm # Stop the clock

print(rf_random2)

rf_random2$finalModel
varImpPlot(rf_random2$finalModel, main = "Random Forest Model: \n Variable Importance")

# Predict train
set.seed(123)
rf_random2.pred <- predict(rf_random2, newdata = train,
                           type = "prob")[,2]

# Plot ROC curve
set.seed(123)
rf_random2.roc <- plot.roc(train$y, rf_random2.pred)
rf_random2.auc <- rf_random2.roc$auc
rf_random2.auc # Area under the curve: 1
```

```
par(pty = "s") # "s" generates a square plotting region
plot(rf_random2.roc, col = "steelblue", main = "ROC Curve for Random Forest Model")
par(pty = "m") # "m" generates the maximal plotting region

# Predict train for confusion matrix
set.seed(123)
rf_random2.pred2 <- predict(rf_random2, newdata = train) # no type = "prob"
set.seed(123)
rf_random2.cmat <- confusionMatrix(rf_random2.pred2, train$y)
rf_random2.cmat

# Predict test
set.seed(123)
rf_random2.pred.test <- predict(rf_random2, newdata = test)

set.seed(123)
rf_random2.cmat.test <- confusionMatrix(rf_random2.pred.test, test$y)
rf_random2.cmat.test

############################################################
# End
############################################################
```