Assignment #3

James R. Herbick

Introduction

Overview / Purpose

This assignment involves analyzing frequency counts of certain words and characters from individual emails for the purpose of developing a model to predict spam. The observations for this assignment were assembled with a mix of spam emails and non-spam emails. In addition, they represent some emails to a particular person, therefore, this exercise is more of a personalized spam filter than a general spam filter. The purpose of this assignment is to develop a model to classify email observations as either spam or not spam.

Before beginning the model-building process, I applied a series of variable-selection techniques. These techniques help identify independent variables that may have the strongest relationship with the dependent variable. In addition, my exploratory data analysis (EDA) looked at frequencies of values, outliers, and model output to uncover important relationships between variables. Finally, I explored multiple modeling methods and determined the best model for predicting whether or not an email is spam. The modeling techniques that I explored are: logistic regression, classification trees, support vector machines, and a random forest.

Modeling Problem Statement

The modeling problem in this assignment is a pure prediction problem in a binary classification setting. However, throughout the model-building process, it was important to understand the relationship of certain variables to the dependent variable, SPAM. The dependent variable in this assignment is a binary categorical variable indicating whether an email is spam or not. Therefore, this modeling problem is a binary classification problem. Modeling techniques appropriate for binary classification were explored. The predictive modeling techniques used in this assignment were: logistic regression, classification trees, support vector machines, and a random forest. Finally, completion of this assignment will provide a predictive modeling framework for working through a binary classification modeling problem.

Data

General Description

The dataset for this assignment includes 4,601 observations and 57 input variables. Each observation contains the frequency counts of certain words and characters from a single email. All of the 57 input variables are continuous, and represent these counts or frequencies. These frequencies are not integer values, they contain decimal points, so they truly are continuous. The dependent variable, SPAM, is a

binary categorical variable that represents whether or not an email is spam. Figure 1 is a table of all the variables in the dataset with summary information about each. In my initial pass on this data, no transformations were performed.

FIGURE 1: Data elements

	Variable Name	Data Type	Original/ Derived	Description
	wf_make wf_address wf_all wf_3d wf_our wf_over wf_remove wf_internet wf_order wf_mail wf_receive wf_will wf_people wf_report wf_addresses wf_free wf_business wf_email wf_you wf_credit wf_your wf_font wf_000 wf_money wf_hp wf_hpl wf_george wf_650 wf_lab wf_labs wf_telnet wf_857 wf_data wf_415 wf_85 wf_technology wf_1999 wf_parts	Continuous	Original	All continuous, independent variables, that represent counts or frequencies of either words or symbols within a single email. These frequencies are not integer values, but decimal and therefore are truly continuous. Those variables beginning with a wf_ are word frequency counts. Those variables beginning with cf_ are symbol or character counts. For the character counts, the character has been spelled out as a part of the variable name.

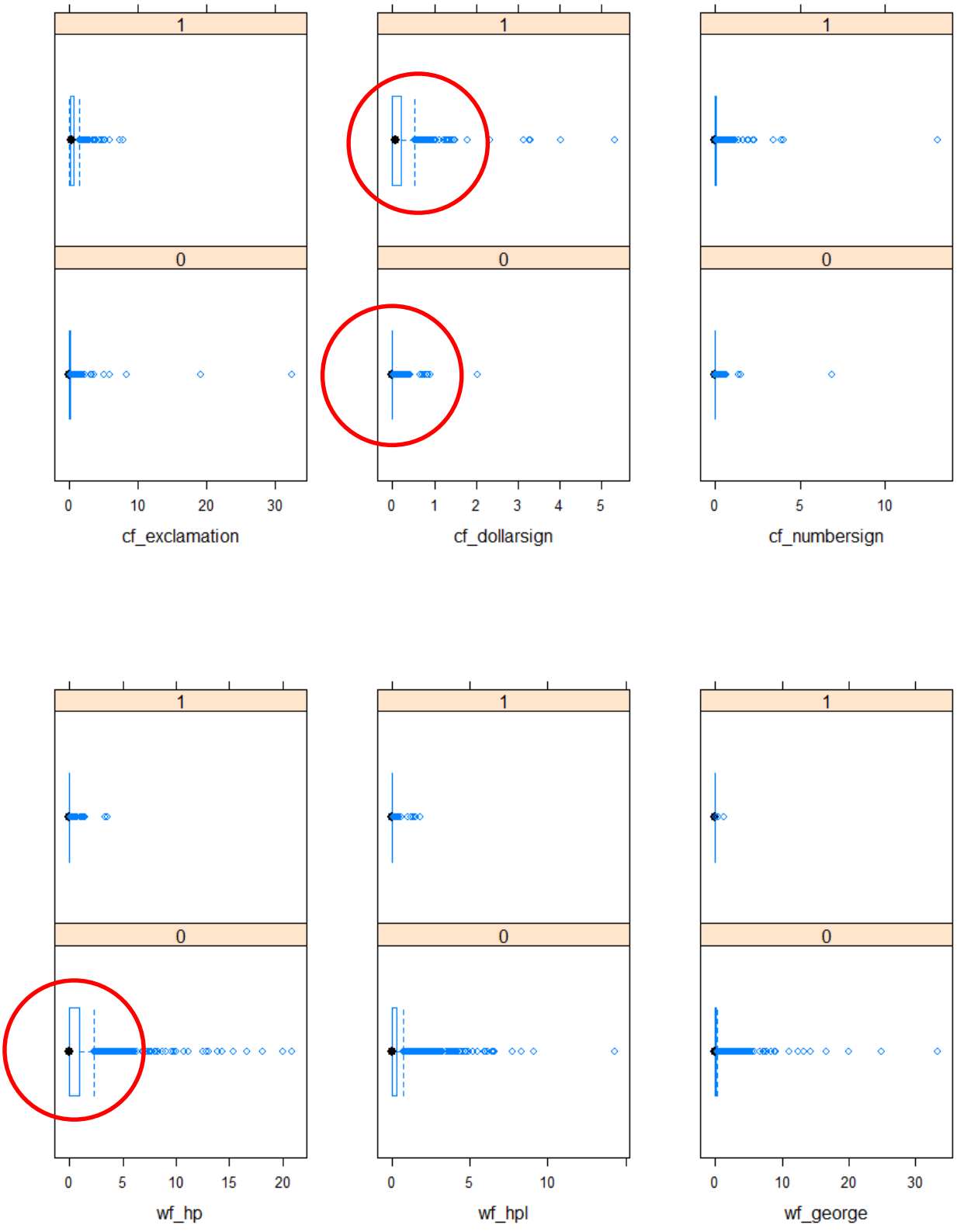
	wf_pm wf_direct wf_cs wf_meeting wf_original wf_project wf_re wf_edu wf_table wf_conference cf_semicolon cf_parens cf_sqbracket cf_exclamation cf_dollarsign cf_numbersign average longest total			
	SPAM	Binary, Categorical	Original	The dependent variable. Denotes whether a specific email (observation) is spam or not.

Data Quality Check

As mentioned in the introduction, the first step in any modeling project is to perform a data quality check. The purpose of this step of the model-building process is to identify any missing values or outliers in the data. An additional objective of the data quality check is to identify any potential issues in the data that could affect the model-building process. There are no missing values in the data. However, there are several word or character frequencies that are zero, in fact quite a few. As I learned in previous coursework, the absence of a variable may actually be predictive. Therefore, the absence of certain words or characters may actually help predict whether an email is spam or not.

Figure 2 shows a select number of boxplots for a few independent variables, categorized by the dependent variable, SPAM. In the first row of boxplots, one can see that non-zero counts for the dollar sign variable are a bit more prevalent for spam, as opposed to not spam. The 25^{th}, 50^{th}, and 75^{th} percentiles shown in the collapsed box for dollar sign with SPAM = 0, indicate that a lack of a dollar sign in an email, may suggest that the email is not spam. In contrast, on the second row of boxplots, one sees a slightly larger non-zero count of 'hp' when an email is not spam. These insights lead us to believe that a dollar sign is more likely to indicate spam, while higher counts of 'hp' indicate an email that is not spam.

FIGURE 2: Outliers by SPAM



Another important aspect of the data quality check is investigating the distributions of the data. Figure 3 shows histograms of the observations with regard to exclamation, dollarsign, and the numbersign symbols.

Figure 3: Independent variable histograms (exclamation, dollarsign, and numbersign)

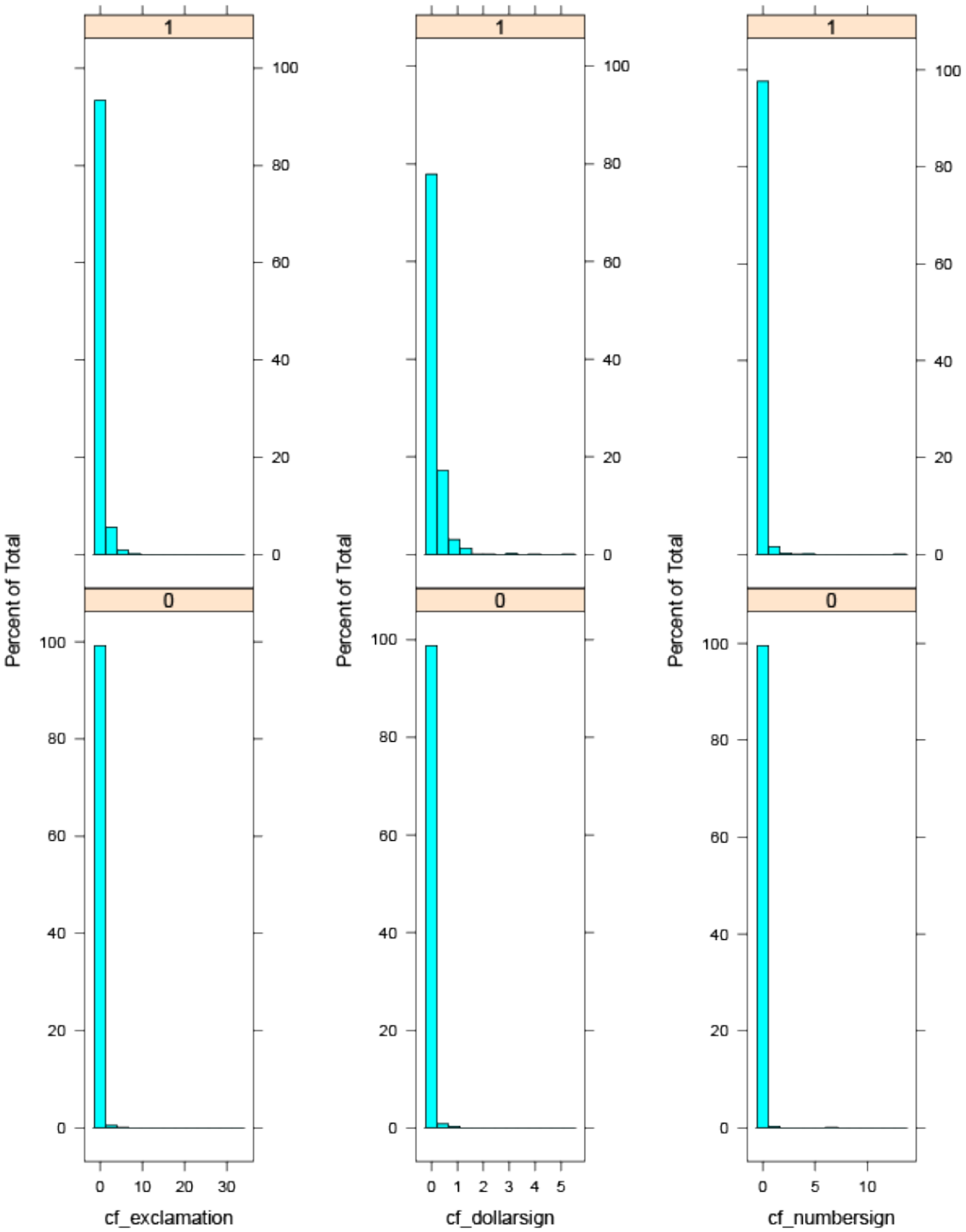
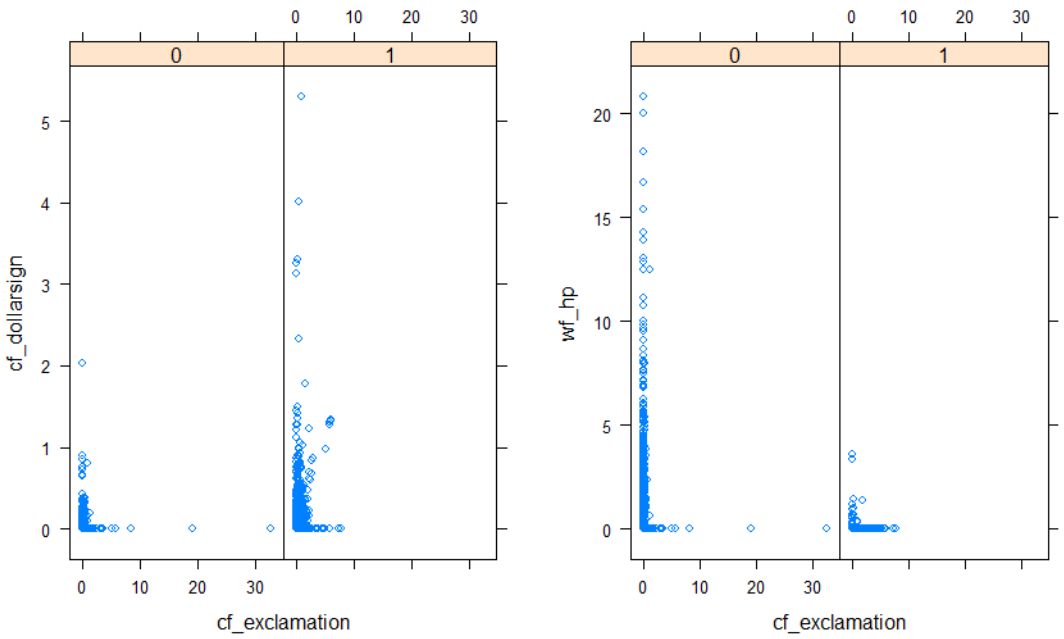


Figure 3 shows that across all of these variables, there is definitely a zero-inflated situation. The counts of zero greatly outweigh other frequencies. If these counts were truly integer, or count-based, then potentially zero inflated Poisson techniques could be used. However, these counts include decimal points and are truly continuous.

Exploratory Data Analysis (EDA)

After performing the data quality check, I proceeded to explore the relationships between the dependent and independent variables. I was sure to run EDA on the training data, after performing a 70% / 30% train / test split. Figure 4 shows a couple of select scatterplots depicting relationships between independent variables, grouped by dependent variable values. In the left panel of figure 4, one sees that there is a slight positive correlation between cf_exclamation and cf_dollarsign when SPAM equals 1. Therefore, both cf_exclamation and cf_dollarsign may be related to spam emails. On the right side of figure 4, one can see that when SPAM equals 0, there are very few cf_exclamation values and a series of wf_hp values. This may imply a negative correlation between these independent variables. That is, when there are few cf_exclamation counts and a large number of wf_hp counts, then the email is likely not be spam.

Figure 4: Scatterplot matrices for select variables by SPAM

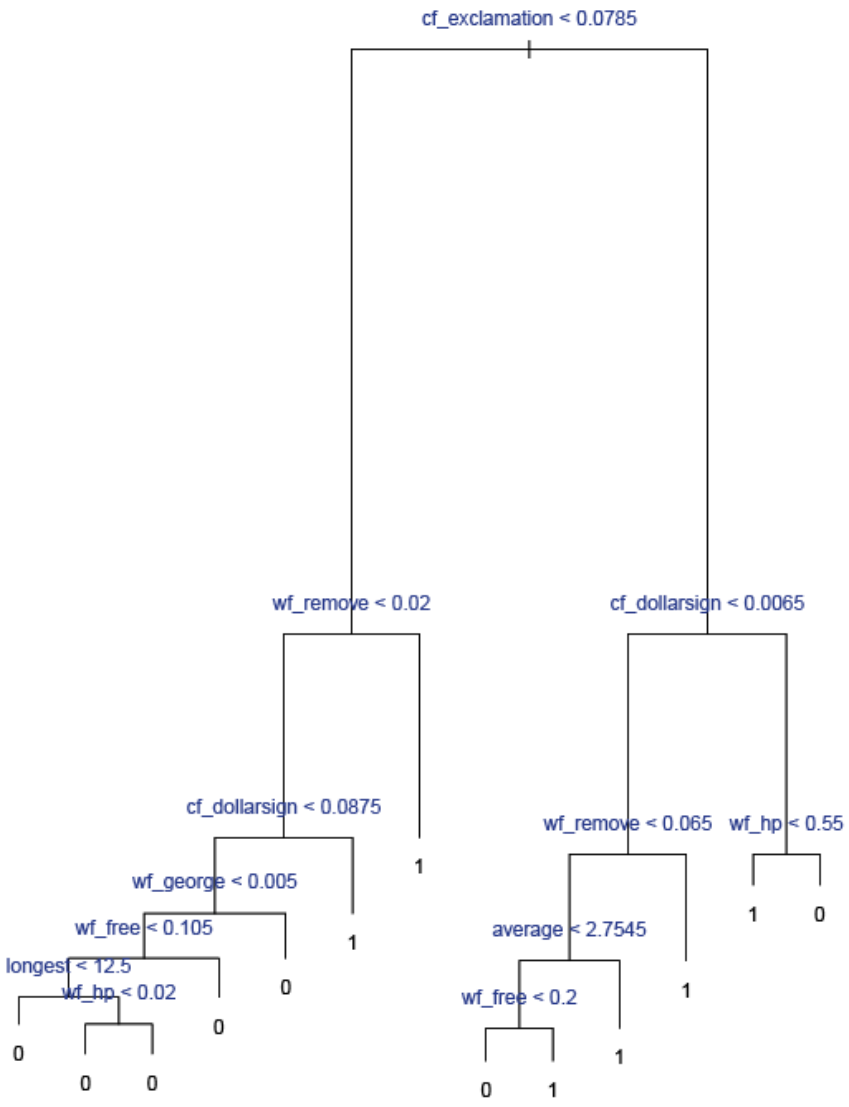


Model-Based EDA

As a final step in my EDA, I utilized a classification tree to confirm previous insights and to gain additional insights into those independent variables that may contribute significantly to whether or not

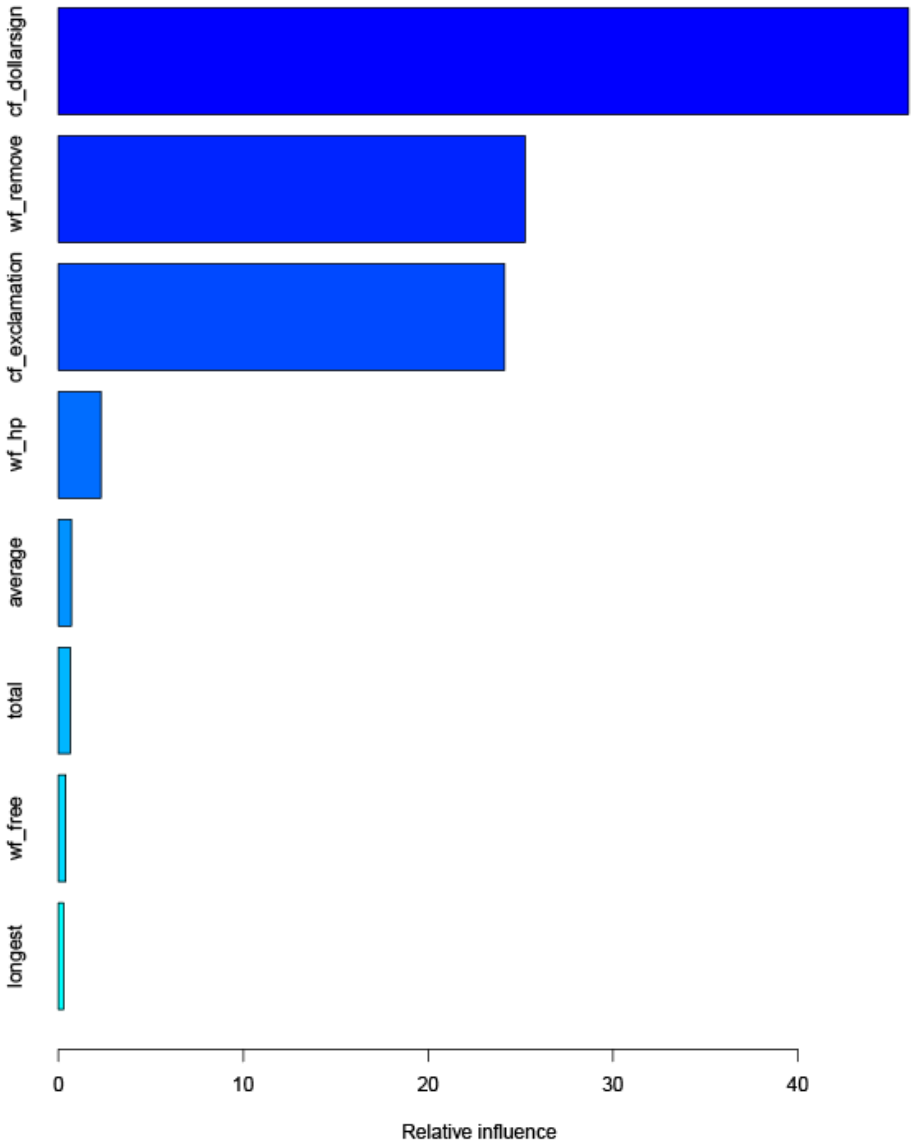
an email is spam. Figure 7 shows the graphical depiction of an unpruned classification tree. According to the classification tree, the exclamation point is the most predictive independent variable of the 57. Those observations that have greater than .078 exclamation points seem much more likely to be classified as spam. This is observed by following the right-hand side of the tree. The word remove and the dollar sign character also have significant predictive power. It appears that higher frequencies of both cf_dollarsign and wf_remove, the more likely an email is spam. Higher frequencies of wf_george, wf_hp, and wf_free seem to indicate email that is not spam.

FIGURE 7: EDA Classification tree



In addition to a basic classification tree, I ran a quick set of gradient boosted trees to see what the variable importance results were for the top variables. Figure 8 shows the variable importance of the top eight independent variables according to the gradient boosting procedure. Again, cf_dollarsign, cf_exclamation, wf_hp, and wf_remove are shown to have the most predictive power.

FIGURE 8: EDA Gradient Boosted Trees



Model Building

Variable Selection Techniques

I utilized a series of variable selection techniques to examine up to a 25-variable model. These techniques included: forward, backward, and stepwise variable selection. Best subset selection was not performed due to the number of variables involved and the large computational impact running that procedure would have. Figure 10 shows the results of the variable-selection techniques for the top ten variables chosen. As you can see in figure 10, the forward, backward, and stepwise variable selection techniques all selected the same variables. In addition, cf_exclamation, cf_dollarsign, and wf_hp all were chosen later in the variable-selection process than the original EDA would suggest.

Figure 10: Top 10 Variables by variable selection techniques

Independent Variable	Forward	Backward	Stepwise
cf_exclamation	8	8	8
cf_dollarsign	6	6	6
wf_hp	9	9	9
wf_remove	3	3	3
wf_your	1	1	1
wf_000	2	2	2
total	4	4	4
wf_free	5	5	5
wf_our	7	7	7
wf_internet	10	10	10

Naïve Model

To begin the model-building process, I examined a naïve model after using the backward variable selection technique. From the backward variable selection technique, I selected the top 5 variables for the naïve model: wf_your, wf_remove, total, wf_free, and wf_000. Again, these variables represent the top 5 variables chosen by the variable selection techniques. Figure 8 displays the resulting naïve model.

Figure 8: Naïve Logistic regression model

```
SPAM =  -2.0733925229                +
        0.5385371614    * wf_your    +
        6.1142709743    * wf_remove  +
        0.0009430134    * total      +
        1.1892915355    * wf_free    +
        5.7865790516    * wf_000
```

The coefficients of the model in figure 8 suggest that wf_remove and wf_000 have the largest effect on classifying an email as spam. This model had an 84% in-sample precision rate (correct classification rate). The model had an 83% out-of-sample precision rate. These precision rates suggest a good start to a base model. However, the intent is to explore additional modeling techniques in a binary classification setting to settle upon the model with the best predictive power.

Model Suite

The next step in the model-building process was to build various models suitable to a binary classification situation. The model suite included: logistic regression, a classification tree, a support vector machine, and a random forest.

Logistic Regression

The first model that I examined for prediction of SPAM, was a logistic regression model. Based upon the variable selection methods (forward, backward, and stepwise), I selected the top 6 variables from which to build a logistic regression model. In addition, I added the cf_exclamation variable due to the model-based EDA results. The model is shown in figure 11. All of the model's variables were statistically significant.

Figure 11: Logistic regression model

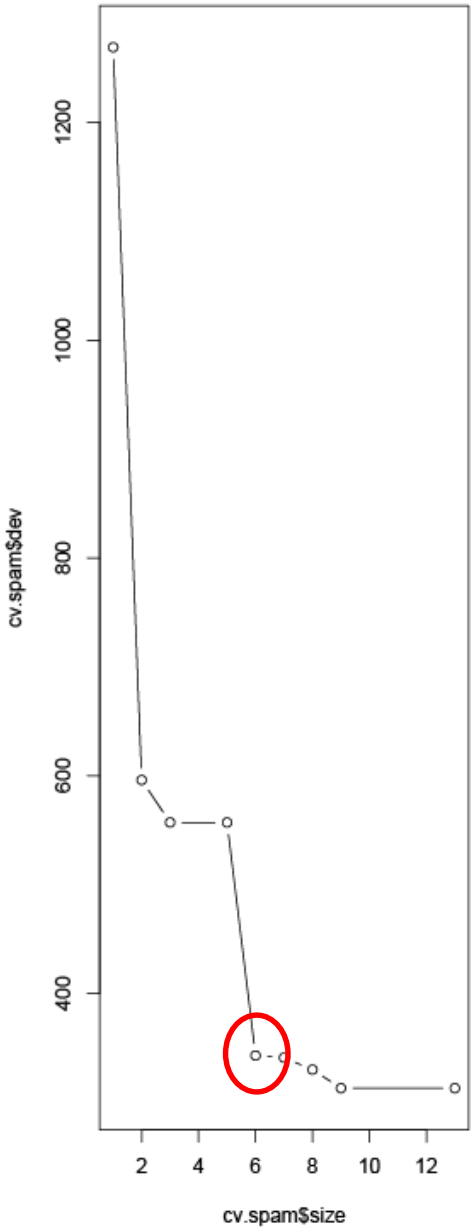
```
SPAM =  -2.4267187                    +
        0.4563175     * wf_your       +
        4.7694496     * wf_000        +
        5.9420761     * wf_remove     +
        0.0008094     * total         +
        0.9995871     * wf_free       +
        6.2682440     * cf_dollarsign +
        1.0092683     * cf_exclamation
```

The coefficients of the model in figure 8 suggest that cf_dollarsign and wf_remove now have the largest effect on classifying an email as spam. This model had an 87% in-sample precision rate (correct classification rate). The model had an 86% out-of-sample precision rate. The addition of cf_dollarsign and cf_exclamation have improved the model performance over that of the naïve model.

Classification Tree

The next modeling technique that I utilized was a classification tree. Figure 12 shows the cross validation error rate plotted by the tree size. Figure 12 shows that a tree size of 6 can minimize the error rate while keeping the tree size small.

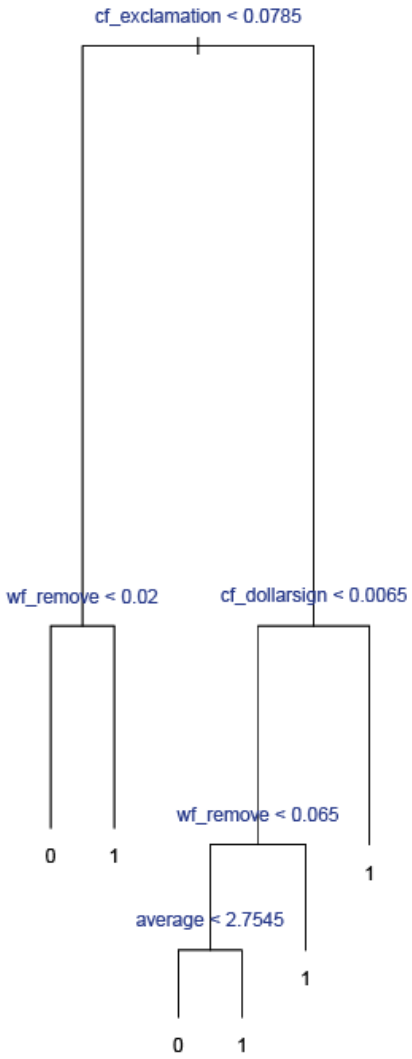
Figure 12: Cross validation error rate by tree size



Therefore, the full classification tree was pruned to a size of six terminal nodes. The pruned tree is shown in figure 13. The pruned classification tree shows that the cf_exclamation variable is the most important in determining whether or not an email is spam. The wf_remove and cf_dollarsign variables are also key variables. The higher the frequency of cf_exclamation, cf_dollarsign and wf_remove, the more likely that an email will be considered spam. These results are consistent with the model-based

EDA that was performed. The pruned classification tree had an 89% precision rate both in and out of sample. This modeling approach gains in predictive accuracy over both of the logistic regression models.

Figure 13: Pruned Classification Tree

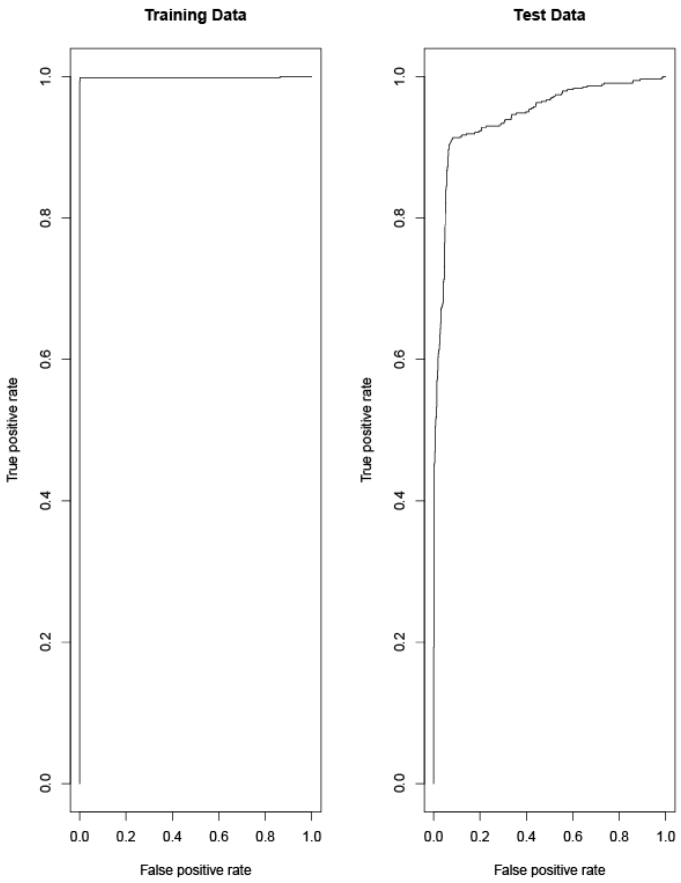


Support Vector Machine (SVM)

The next modeling approach that I attempted was a support vector machine. I began the SVM modeling with a radial kernel, a gamma of 1, and a cost of 0.00005. The type of kernel chosen can allow the SVM to fit a non-linear decision boundary. The use of kernels is important because they allow an expansion of the feature space, while moderating the computational impacts. For these reasons, a radial kernel was chosen. The cost parameter can be thought of as a budget for the amount that the margin can be violated by the observations. This initial setting has a very small budget.

I then optimized the SVM for the best fit. The optimized parameters were a cost of 10 and a gamma of 0.5. The results of the optimized model for both the training and test data are shown in figure 14. The figure shows the ROC curves for the optimized SVM model on both the training and test data. The ROC curve for the training data is almost perfect. The ROC curve for the test data is less desirable. However, the near-perfect fit on the training data suggests that the SVM model most likely is overfitting the data. The SVM's precision in sample was 99%, while the out-of-sample precision was 82%. This out-of-sample performance was the worst of all the modeling techniques.

Figure 14: SVM ROC curves

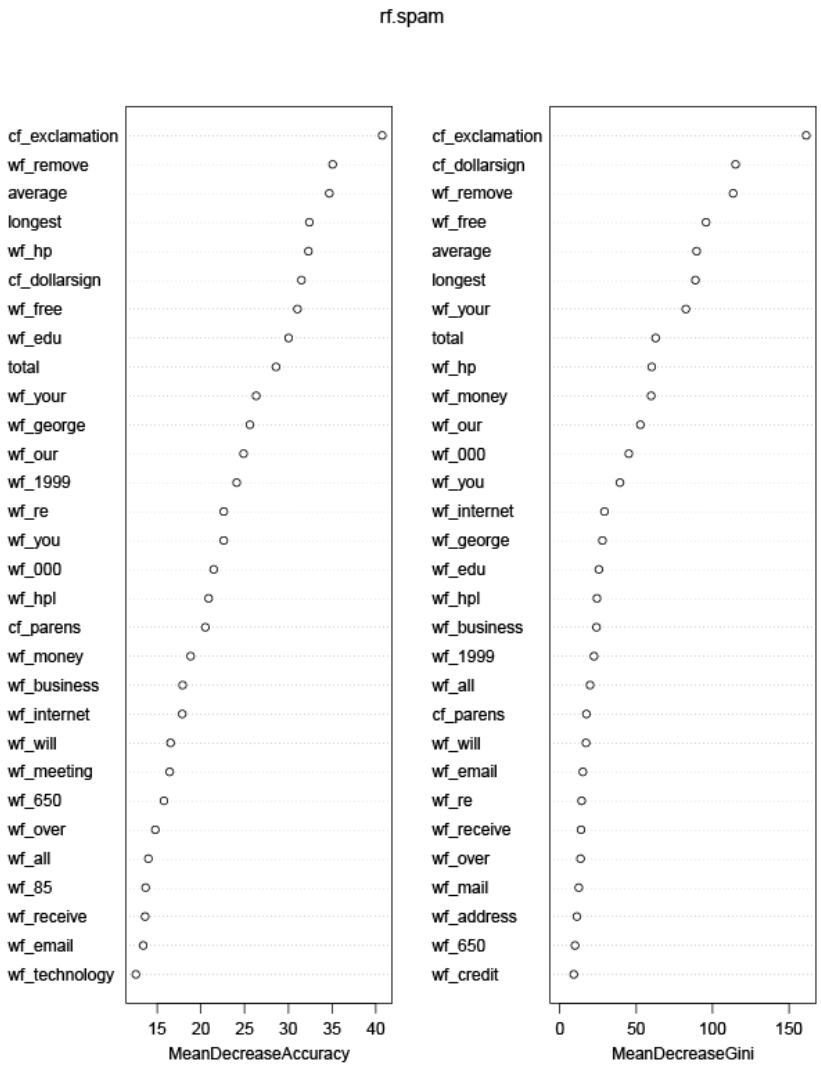


Random Forest

A random forest modeling approach not only includes the benefits of bagging, which uses bootstrapped samples, but it also decorrelates the trees by using a random subset of independent variables for determining each split. The default behavior for the random forest method is to select $\sqrt{p}$ as the number of variables to consider at each split. In the spam data case, this would equate to 7.5 variables. I instead lowered that value to 5. Therefore, at each split, a maximum of 5 random variables could be considered.

Figure 13 shows the importance of variables as determined by the random forest modeling process. Variable importance is displayed by the percent increase in MSE when a particular variable is removed from the model. Variable importance is also shown as the increase in node purity provided by the variable. We see that cf_exclamation tops the list of variables as far as importance goes. cf_dollarsign and wf_remove are the next most important variables from a purity perspective.

Figure 13: Random forest variables



These variable-importance results confirm our analysis in the model-based EDA process.

Conclusions

Figure 14 summarizes the in-sample and out-of-sample model precision (correct classification rate). Although the naïve logistic regression model was only the 4th best of all the models, its performance was not that far behind some of the other techniques. This reflects the importance of investigation to the most predictive variables during the EDA process. By looking at the EDA results holistically, I was able to improve over the naïve model's performance, slightly with another logistic regression model. The classification tree approach had better predictive accuracy still. Tree-based approaches are resilient to outliers and missing data. Therefore, they may have been able to handle the zero-inflated data a bit better than straight logistic regression.

The results also present a lesson of caution. It appears that the SVM model overfit the training data, even though an automated procedure was used to optimize the model. Because of this overfitting, the SVM performance out-of-sample was the worst of all the models. Finally, the random forest approach was the most accurate and its performance differed little between in and out of sample. The fact that random forests use boostraped training samples and a random subset of the independent variables throughout the training process, demonstrates the power of sampling and variation in the model-building process.

Figure 14: Model Results

Model	In-Sample Precision	Out-of-Sample Precision	Model Rank
Naïve Logistic Regression	84%	83%	4
Logistic Regression	87%	86%	3
Classification Tree	89%	89%	2
Support Vector Machine	99%	82%	5
Random Forest	95%	95%	1

Appendix

R Code

This section contains only the model building R code.

```
# --------------------------------
# Naive Logistic Regression Model
# --------------------------------


# -------------------
# Logistic Regression
# -------------------

# fit a logistic regression model to the data
glm.fit=glm(SPAM~wf_your+wf_remove+total+wf_free+wf_000, data=spam.train, family=binomial)
summary(glm.fit)

# Look at the logistic regression model coefficients
coef(glm.fit)
summary(glm.fit)$coef

# Look at the p values for the coefficients
summary(glm.fit)$coef[,4]

# Make predictions... type="response" ensures probabilities are returned.
# with no data argument, this predicts on the training data
glm.probs=predict(glm.fit,type="response")

# Examine first 10 probabilities
glm.probs[1:10]
contrasts(spam.train$SPAM)  # lets you see how dummy variables are coded in R.

# Convert probabilities to an Up or Down... > .5 = Up
# rep command repeats Down 1250 times....
glm.pred=rep("0",3229)
glm.pred[glm.probs>.5]="1"

# Examine frequencies
table(glm.pred,spam.train$SPAM)  # confusion matrix
(1854+870)/3229  # Calculate overall prediction accuracy

mean(glm.pred==spam.train$SPAM)  # fraction of correct predictions


# Make predictions on the test set
glm.probs=predict(glm.fit, spam.test, type="response")

# Change predictions to Ups and Downs
glm.pred=rep("0",1372)
```

```
glm.pred[glm.probs>.5]="1"

# Confusion matrix and percentages of accurate predictions
table(glm.pred, spam.test$SPAM)
mean(glm.pred==spam.test$SPAM)
mean(glm.pred!=spam.test$SPAM)




# ---------------
# Fit model suite
# ---------------


# -------------------
# Logistic Regression
# -------------------


# fit a logistic regression model to the data
glm.fit.2=glm(SPAM~wf_your++wf_000+wf_remove+total+wf_free+cf_dollarsign, data=spam.train,
family=binomial)
summary(glm.fit.2)

# try adding in exclamation
glm.fit.3=glm(SPAM~wf_your++wf_000+wf_remove+total+wf_free+cf_dollarsign+cf_exclamation,
data=spam.train, family=binomial)
summary(glm.fit.3)



# Look at the logistic regression model coefficients
coef(glm.fit.2)
summary(glm.fit.2)$coef

# Look at the p values for the coefficients
summary(glm.fit.2)$coef[,4]

# Make predictions... type="response" ensures probabilities are returned.
# with no data argument, this predicts on the training data
glm.probs2=predict(glm.fit.2,type="response")

# Make predictions... type="response" ensures probabilities are returned.
# with no data argument, this predicts on the training data
glm.probs3=predict(glm.fit.3,type="response")



# Examine first 10 probabilities
glm.probs2[1:10]
contrasts(spam.train$SPAM)  # lets you see how dummy variables are coded in R.
```

```
# Convert probabilities to an Up or Down... > .5 = Up
# rep command repeats Down 1250 times....
glm.pred2=rep("0",3229)
glm.pred2[glm.probs2>.5]="1"

# Examine frequencies
table(glm.pred2,spam.train$SPAM)  # confusion matrix
(1854+931)/3229  # Calculate overall prediction accuracy

mean(glm.pred2==spam.train$SPAM)  # fraction of correct predictions


# Convert probabilities to an Up or Down... > .5 = Up
# rep command repeats Down 1250 times....
glm.pred3=rep("0",3229)
glm.pred3[glm.probs3>.5]="1"

# Examine frequencies
table(glm.pred3,spam.train$SPAM)  # confusion matrix
# (1854+931)/3229  # Calculate overall prediction accuracy

mean(glm.pred3==spam.train$SPAM)  # fraction of correct predictions




# Make predictions on the test set
glm.probs2.test=predict(glm.fit.2, spam.test, type="response")

# Change predictions to Ups and Downs
glm.pred2.test=rep("0",1372)
glm.pred2.test[glm.probs2.test>.5]="1"

# Confusion matrix and percentages of accurate predictions
table(glm.pred2.test, spam.test$SPAM)
mean(glm.pred2.test==spam.test$SPAM)
mean(glm.pred2.test!=spam.test$SPAM)



# Make predictions on the test set
glm.probs3.test=predict(glm.fit.3, spam.test, type="response")

# Change predictions to Ups and Downs
glm.pred3.test=rep("0",1372)
glm.pred3.test[glm.probs3.test>.5]="1"

# Confusion matrix and percentages of accurate predictions
table(glm.pred3.test, spam.test$SPAM)
mean(glm.pred3.test==spam.test$SPAM)
mean(glm.pred3.test!=spam.test$SPAM)
```

```
# --------------------
# Classification Tree
# --------------------


library(tree)
library(ISLR)


# Fit a classification tree
tree.spam.2=tree(SPAM~., spam.train)
summary(tree.spam.2)

# Display the tree graphically
plot(tree.spam.2)
text(tree.spam.2,pretty=0)
tree.spam.2


# Test model on test dataset
tree.pred.2=predict(tree.spam.2, spam.test, type="class")  # class means return the actual classification prediction
table(tree.pred.2, spam.test$SPAM)
(750+470)/1372


# Prune the tree
set.seed(3)
cv.spam=cv.tree(tree.spam.2, FUN=prune.misclass) # FUN=prune.misclass here states to use the classification
error rate  to guide pruning
names(cv.spam)                          # default is for deviance to guide....
cv.spam

# plot the cross validation error rate by tree size and k value
par(mfrow=c(1,2))
plot(cv.spam$size, cv.spam$dev,type="b")
plot(cv.spam$k, cv.spam$dev,type="b")

# Prune to the best model based on graphs
# The 6-node tree was the best.
prune.spam=prune.misclass(tree.spam.2, best=6)
plot(prune.spam)
text(prune.spam, pretty=0)


# check the training set performance of the 6-node tree
tree.pred.pruned.train=predict(prune.spam, spam.train, type="class")
table(tree.pred.pruned.train, spam.train$SPAM)
(776+432)/1372
```

```
# Check test set performance of the 6-node tree
tree.pred.pruned=predict(prune.spam, spam.test, type="class")
table(tree.pred.pruned, spam.test$SPAM)
(776+432)/1372



# -------------
# SVM
# -------------

library(e1071)


# Fit support vector machine using a radial kernel, gamma = 1
svmfit=svm(SPAM~., data=spam.train, kernel="radial",  gamma=1, cost=1)
plot(svmfit, spam.train) # not working

# obtain info about the svm fit
summary(svmfit)

# Increase the cost... risk of overfitting the data
svmfit=svm(y~., data=dat[train,], kernel="radial",gamma=1,cost=1e5)
plot(svmfit,dat[train,])

# Use cross validation to choose the best value of gamma and cost
set.seed(1)
tune.out=tune(svm, SPAM~., data=spam.train, kernel="radial",
ranges=list(cost=c(0.1,1,10,100,1000),gamma=c(0.5,1,2,3,4)))
summary(tune.out)


# train
table(true=spam.train$SPAM, pred=predict(tune.out$best.model, newx=spam.train))

# test
bestmod <- tune.out$best.model
p <- predict(bestmod, spam.test)
table(true=spam.test$SPAM, pred=p)

# ----------
# ROC Curves
# ----------

library(ROCR)

# Create ROC plotting function
rocplot=function(pred, truth, ...){
   predob = prediction(pred, truth)
   perf = performance(predob, "tpr", "fpr")
   plot(perf,...)}
```

```
# Use decision.values=TRUE to get fitted values from SVM
svmfit.opt=svm(SPAM~., data=spam.train, kernel="radial",gamma=.5, cost=10,decision.values=T)
fitted=attributes(predict(svmfit.opt, spam.train, decision.values=TRUE))$decision.values

# Produce the roc plot
par(mfrow=c(1,2))
rocplot(fitted, spam.train$SPAM,main="Training Data")

# Increase gamma for better fit
svmfit.flex=svm(y~., data=dat[train,], kernel="radial",gamma=50, cost=1, decision.values=T)
fitted=attributes(predict(svmfit.flex,dat[train,],decision.values=T))$decision.values
rocplot(fitted,dat[train,"y"],add=T,col="red")

# Most interested in ROC curve on the test data
fitted=attributes(predict(svmfit.opt, spam.test, decision.values=T))$decision.values
rocplot(fitted, spam.test$SPAM, main="Test Data")

fitted=attributes(predict(svmfit.flex,dat[-train,],decision.values=T))$decision.values
rocplot(fitted,dat[-train,"y"],add=T,col="red")




# -------------
# Random Forest
# -------------

library(randomForest)

# Random forest using 6 random variables
set.seed(1)
rf.spam=randomForest(SPAM~., data=spam.train, mtry=5, importance=TRUE)

# how do I build my confusion matrix????
table(rf.spam$predicted, spam.train$SPAM)
(1900+1175)/3229
# mean(rf.spam$mse)


yhat.rf.spam = predict(rf.spam, newdata=spam.test)
table(yhat.rf.spam, spam.test$SPAM)
(802+496)/1372

# mean((test$logprice-yhat.rf)^2)

# View the importance of each variable
importance(rf.spam)
varImpPlot(rf.spam)  # Plot the importance measures
```