Email Classification Report (spam or not)

Kumar (K.J.) Gandhi
Justin Fell

Introduction & Literature Review

Data source information:

The dataset was donated to the UCI Machine Learning Repository by Hewlett-Packard Labs in 1999. It contains 4601 emails. Here is a link: Spam

Introduction of the data:

The "spam" concept is diverse: advertisements for products/websites, make money fast schemes, chain letters, pornography. This collection of spam emails came from postmaster and individuals who had filed spam while the collection of non-spam emails came from filed work and personal emails. The dataset contains 4601 emails, and 58 variables in total. About 39% of the emails are spam, while the rest are non-spam.

Dataset Attributes:

The last column of 'spambase.data' denotes whether the email was considered spam (1) or not (0), i.e. unsolicited commercial e-mail. Most of the attributes indicate whether a particular word or character was frequently occuring in the email. The run-length attributes (55-57) measure the length of sequences of consecutive capital letters. For the statistical measures of each attribute, see the end of this file. Here are the definitions of the attributes:

48 continuous real [0,100] attributes of type word_freq_WORD = percentage of words in the email that match WORD, i.e. 100 * (number of times the WORD appears in the email) / total number of words in email. A "word" in this case is any string of alphanumeric characters bounded by non-alphanumeric characters or end-of-string.

6 continuous real [0,100] attributes of type char_freq_CHAR] = percentage of characters in the email that match CHAR, i.e. 100 * (number of CHAR occurrences) / total characters in email

1 continuous real [1,…] attribute of type capital_run_length_average = average length of uninterrupted sequences of capital letters

1 continuous integer [1,…] attribute of type capital_run_length_longest = length of longest uninterrupted sequence of capital letters

1 continuous integer [1,…] attribute of type capital_run_length_total = sum of length of uninterrupted sequences of capital letters = total number of capital letters in the email

1 nominal {0,1} class attribute of type spam = denotes whether the email was considered spam (1) or not (0), i.e. unsolicited commercial e-mail.

Scientific Goal:

Our goal is to classify each email to determine if it is spam or not using various machine learning techniques such as SVM, Logistic Regression, Random Forest. We will also test how other techniques perform such as KNN, CART Model, and LDA. We will find the best classification algorithm based on the accuracy it performs over the dataset.

Literature Review:

There are many statisticians and researchers who have used this dataset to test the performance of machine learning methods. In this analysis by Mathias Schilling, the author compares three machine learning techniques on the spam dataset including Random Forest, KNN, and SVM. He builds testing and training sets from 1000 of the emails where 70% are training data and the rest are testing. In his conclusion, he measures the performance of each algorithm. He finds that Random Forest performs the best with accuracy of 92.31%, while KNN and SVM perform about the same at 88.96%. He notes that he did not spend time tuning the algorithms, so Random Forest is only the most accurate at an initial level.

Load Packages

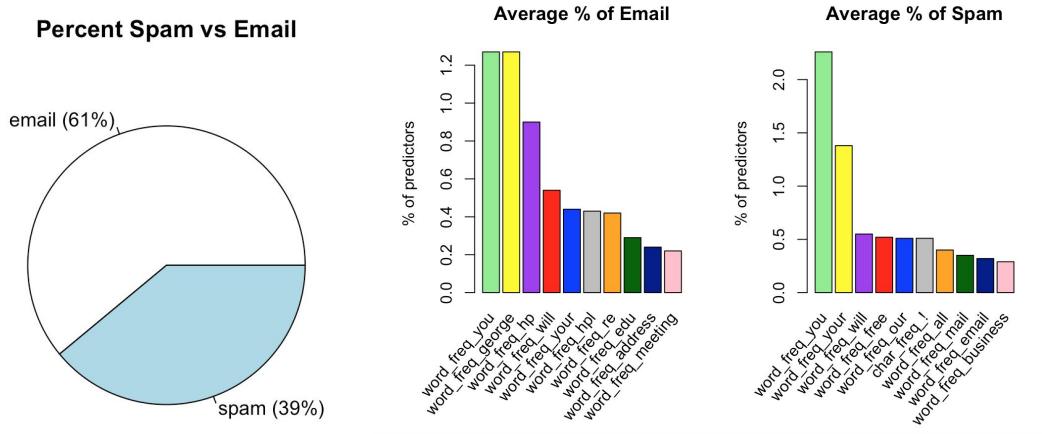
```{r}
library("e1071")
library("MASS")
library("gridBase")
library("grid")
library("class")
library("caret")
library("randomForest")
library("rpart.plot")
```

Summary Statistics & Data Visualization

The first thing that we did with our data was generated summary statistics, as well as generate a few manipulations to see possible trends in our data before performing additional analysis. The spambase dataset that we selected was based off emails, and had a column that marked each row of data as spam or email (non-spam). The many variables in our dataset we found could be used to classify whether an email was spam or email, and we could see that most of our future analysis would be trying to find the best

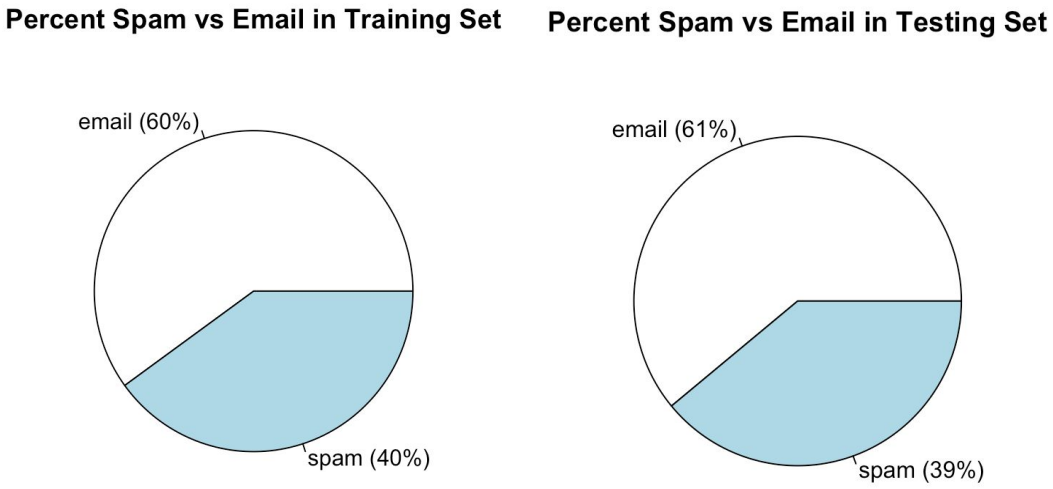
method to create our own spam filter. Our data included 48 continuous variables, which represented the percentage of all words in the email that matches a specific words, such as ‘you’, ‘free’, and ‘people’. There were also six continuous variables that represented the percentage of all characters in the email that matches specific characters, such as ‘$’, ‘%’, and ‘#’. Lastly, there were three variables that related to strings of capital letters, to detect for long sequences of all capital letters in the email. We started by creating some basic summaries of the data, such as finding what percent of emails were marked as spam and what characters and words appeared at the highest frequencies within spam and non-spam emails. Below is the code and plots we generated.

```{r}
x = c(61,39)
labels <- c("email", "spam")
pie(x,labels, main="Percent Spam vs Email")
library(gridBase)
library(grid)
data$spam <- with(data, ifelse(data$spam == 1, "spam", "email"))
dataset.email <- sapply(data[which(data$spam == "email"), 1:54], function(x) ifelse(is.numeric(x),
 round(mean(x), 2), NA))
dataset.spam <- sapply(data[which(data$spam == "spam"), 1:54], function(x) ifelse(is.numeric(x),
 round(mean(x), 2), NA))

dataset.email.order <- dataset.email[order(-dataset.email)[1:10]]
dataset.spam.order <- dataset.spam[order(-dataset.spam)[1:10]]

par(mfrow = c(1, 2))
par(mar = c(8, 4, 4, 2) + 0.1)  # increase y-axis margin.
plot <- barplot(dataset.email.order, col = c( "lightgreen","yellow", "purple", "red", "blue", "grey",
"orange", "darkgreen", "darkblue", "pink"), main = "Average % of Email",
  names.arg = "", ylab = "% of predictors")
# text(x=plot,y=dataset.email.order-0.1, labels=dataset.email.order,
# cex=0.6)
vps <- baseViewports()
pushViewport(vps$inner, vps$figure, vps$plot)
grid.text(names(dataset.email.order), x = unit(plot, "native"), y = unit(-1,
  "lines"), just = "right", rot = 50)
popViewport(3)

plot <- barplot(dataset.spam.order, col = c( "lightgreen","yellow", "purple", "red", "blue", "grey",
"orange", "darkgreen", "darkblue", "pink"), main = "Average % of Spam",
names.arg = "", ylab = "% of predictors")
# text(x=plot,y=dataset.spam.order-0.1, labels=dataset.spam.order,
# cex=0.6)
vps <- baseViewports()
pushViewport(vps$inner, vps$figure, vps$plot)
grid.text(names(dataset.spam.order), x = unit(plot, "native"), y = unit(-1,
  "lines"), just = "right", rot = 50)
popViewport(3)
```



Approximately 39% of the spambase dataset was marked as spam, with the other 61% being email. The exact numbers were 2788 non-spam emails and 1813 spam emails. We also noticed that the frequency of words and characters was different between the two, something that was early encouragement that they could be good predictors in a model detecting spam. In regular emails, words such as 'you', 'address', 'edu', and 'meeting' appeared often, whereas in spam emails, while 'you' again was a common word, 'your' was used much more often along with 'free', '!', and 'our'. To prepare for implementing our methods such as KNN, SVM, and random forest, we had to split our data up into a testing and a training set. We elected to put 66% of our data in the training set, with the remaining 33% going into the test set. The percentages of spam and non-spam in each were very similar to the spambase dataset as a whole, which you can see below.

```{r}
set.seed(9406)
index <- 1:nrow(data)
trainIndex <- sample(index, trunc(length(index) * 0.666666666666667))
data.train <- data[trainIndex, ]
print(paste0("Percentage: ", round((nrow(data.train)/nrow(data)) * 100,
  2), " %"))
data.test <- data[-trainIndex, ]
print(paste0("Percentage: ", round((nrow(data.test)/nrow(data)) * 100,
   2), " %"))
table(data.train$spam)
table(data.test$spam)

x = c(60,40)
labels <- c("email", "spam")
pie(x,labels, main="Percent Spam vs Email in Training Set")

x = c(61,39)
labels <- c("email", "spam")
pie(x,labels, main="Percent Spam vs Email in Testing Set")
```



As a group, we felt that the data was well formatted for us to perform visualization and summary statistics on. The statistics and plots we generated were accurate and represented the data precisely. They also were helpful for us in terms of being able to see differences in our variables between the spam and non-spam emails in our data, which was a positive sign moving into developing methods and models for our spam detector. Our models and model analysis will present more visualization later, but not of the general data and summary statistics. Looking at the summary statistics and overall data through R outputs and visualizations allowed us to know how to use the spambase dataset to create a model predicting spam or non-spam emails using a variety of methods.

Proposed Analysis

K-Nearest Neighbors Method

One method that we knew from the start may not be the best for our data but wanted to try anyway is the *k*-nearest-neighbors, or KNN. Since our output variable of 'is_spam' is binary, resulting in either regular email (0) or spam (1), performing KNN was limited to generating confusion matrices rather than plots of a linear or geometric model. After converting the output variable data into a binary output of 0 and 1 from a string output of 'email' and 'spam', I performed knn using the training and test sets and generated a couple of confusion matrices with multiple k values. The k-values that I selected to use were k=1, k=5, k=15, and k=50. The code and results are seen below.

```{r}
# knn predicted vs actual k = 1
data.knn<-knn(train=data.train, test=data.test, cl = data.train$spam, k = 1)
summary(data.knn)

data.pred.df <- data.frame(temp = data.knn)
data.pred <- with(data.pred.df, ifelse(temp == 1, "spam", "email"))
```

```
data.actual.df <- data.frame(temp = data.test$spam)
data.actual <- with(data.actual.df, ifelse(temp == 1, "spam", "email"))
table(`Actual Class` = data.actual, `Predicted Class` = data.pred)

error.rate.knn <- sum(data.actual != data.pred)/nrow(data.test)
print(paste0("Accuary (Precision): ", 1 - error.rate.knn))

# knn predicted vs actual k = 5
data.knn<-knn(train=data.train, test=data.test, cl = data.train$spam, k = 5)
summary(data.knn)

data.pred.df <- data.frame(temp = data.knn)
data.pred <- with(data.pred.df, ifelse(temp == 1, "spam", "email"))
data.actual.df <- data.frame(temp = data.test$spam)
data.actual <- with(data.actual.df, ifelse(temp == 1, "spam", "email"))
table(`Actual Class` = data.actual, `Predicted Class` = data.pred)

error.rate.knn <- sum(data.actual != data.pred)/nrow(data.test)
print(paste0("Accuary (Precision): ", 1 - error.rate.knn))

# knn predicted vs actual k = 15
data.knn<-knn(train=data.train, test=data.test, cl = data.train$spam, k = 15)
summary(data.knn)

data.pred.df <- data.frame(temp = data.knn)
data.pred <- with(data.pred.df, ifelse(temp == 1, "spam", "email"))
data.actual.df <- data.frame(temp = data.test$spam)
data.actual <- with(data.actual.df, ifelse(temp == 1, "spam", "email"))
table(`Actual Class` = data.actual, `Predicted Class` = data.pred)

error.rate.knn <- sum(data.actual != data.pred)/nrow(data.test)
print(paste0("Accuary (Precision): ", 1 - error.rate.knn))

# knn predicted vs actual k = 50
data.knn<-knn(train=data.train, test=data.test, cl = data.train$spam, k = 50)
summary(data.knn)

data.pred.df <- data.frame(temp = data.knn)
data.pred <- with(data.pred.df, ifelse(temp == 1, "spam", "email"))
data.actual.df <- data.frame(temp = data.test$spam)
data.actual <- with(data.actual.df, ifelse(temp == 1, "spam", "email"))
table(`Actual Class` = data.actual, `Predicted Class` = data.pred)

error.rate.knn <- sum(data.actual != data.pred)/nrow(data.test)
print(paste0("Accuary (Precision): ", 1 - error.rate.knn))
```

```
##   0   1
## 913 621
```

```
##             Predicted Class
## Actual Class email spam
##        email   792  146
##        spam    121  475
```

```
## [1] "Accuary (Precision): 0.825945241199479"
```

```
##   0   1
## 918 616
```

```
##             Predicted Class
## Actual Class email spam
##        email   781  157
##        spam    137  459
```

```
## [1] "Accuary (Precision): 0.808344198174707"
```

```
##   0   1
## 915 619
```

```
##             Predicted Class
## Actual Class email spam
##        email   758  180
##        spam    157  439
```

```
## [1] "Accuary (Precision): 0.780312907431551"
```

```
##   0   1
## 927 607
```

```
##             Predicted Class
## Actual Class email spam
##        email   737  201
##        spam    190  406
```

```
## [1] "Accuary (Precision): 0.745110821382008"
```

k=1 (top left), k=5 (bottom left), k=15 (top right), k=50 (bottom right)

Across all of our methods, the goal was to find a model that would result in the maximum prediction accuracy. To find this for each model, we calculated the error rate and subtracted in from one. After generating the four classification tables for each k, we found that the prediction accuracy for k=1 was 82.5%, for k=5 was 80.8%, for k=15 was 78%, and for k=50 was 74.5%. The larger k we used, the smaller our prediction accuracy was due to a larger error rate. Even at k=1, where our prediction accuracy is maximized, we found we only achieved 82.5% accuracy, which compared to our other models was the lowest by far. From our data initially we knew that KNN likely wouldn't result in the best error rate, and it did not. While the KNN method ultimately did not find us a desirable modeling for detecting spam vs non-spam emails, we better understood how it should be applied to real life data as well as produced a reasonable model for predicting spam. In alternative spam filters and databases, KNN may work better, but for our dataset and project it was not the most optimal choice.

SVM (Support Vector Machine)

SVM works well on high dimension data and only depends on support vectors. SVM is able to do this because it is automatically regularized. The algorithm contains several tuning parameters to optimize its performance and accuracy. Cost or C is the regularization parameter. It controls the tradeoff between misclassifications and the width of the margin. That is, for a larger C there is a smaller margin, the constraints are harder to ignore, and the training points are more correctly classified. On the other hand, for a smaller C there is a larger margin, the constraints are easier to ignore, and the training points will be less correctly classified. The other tuning parameter is gamma. Gamma measures the influence of the training example. For a larger gamma, there will be higher

bias and low variance. This trade-off applies for a smaller gamma where there will be lower bias, and higher variance. To find the best tuning parameters, we implemented the following code:

```{r}
svm_tune <- tune(svm, spam ~ ., data = data.train, kernel = "radial", ranges = list(gamma = c(0.01, 0.05,0.25,1), cost = 10^(-1:2)))
svm_tune$best.parameters
```

We found the best tuning parameters were a gamma value equal to .01 and a cost of 10. When fitting the SVM model and outputting the summary results, we saw the following:

```{r}
model.svm <- svm(spam ~ ., method = "class", cost = 10, data = data.train)
summary(model.svm)
```

```
Call:
svm(formula = spam ~ ., data = data.train, method = "class", cost = 10)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  10
      gamma:  0.01754386

Number of Support Vectors:  760

 ( 397 363 )


Number of Classes:  2

Levels:
 email spam
```

We used a confusion matrix to find the accuracy of the SVM algorithm:

```{r}
prediction.svm <- predict(model.svm, newdata = data.test, type = "class")
table(`Actual Class` = data.test$spam, `Predicted Class` = prediction.svm)
error.rate.svm <- sum(data.test$spam != prediction.svm)/nrow(data.test)
print(paste0("Accuracy (Precision): ", 1 - error.rate.svm))
```

```
             Predicted Class
Actual Class email spam
       email   891   47
       spam     53  543

[1] "Accuracy (Precision): 0.934810951760104"
```

We can see the SVM produces a high accuracy of 93.48% when classifying the emails as spam or not spam,

Logistic Regression

Logistic regression is one of the base statistical methods to classify a dataset with a binary dependent variable. The goal of the logistic regression algorithm is to describe the best model to explain the relation between the dependent variable (spam or not-spam) and the independent variables. In our algorithm, we used logistic regression to classify an observation to the class (spam or not-spam) with the highest probability. We used the cutoff = .5 to minimize the misclassification rate. We ran our model using all of the predictors in the dataset, calculated a confusion matrix and finally, the accuracy.

```{r}
log_model <- glm(spam~., data=data.train, family = "binomial")
pred_full <- predict(log_model, newdata=data.test, type = "response")
pred_full<- ifelse(pred_full > .5, "spam", "email")
(log_confusion <- table(pred_full, data.test$spam))
(log_accuracy = mean(pred_full == data.test$spam))
```

```
pred_full email spam
    email   878   64
    spam     60  532
[1] 0.9191656
```

The logistic regression has a relatively high accuracy of 91.92%, however it uses all of the predictors in the model, so it may be at risk of overfitting.

LDA (Linear Discriminant Analysis)

We implemented the LDA algorithm to classify the data through another machine learning algorithm. LDA attempts to find the best linear combination of the features to separate the data into two or more classifications. The dataset is high dimension, so the results of the LDA were not very precise. Below is the code we used to implement and find the accuracy of our LDA algorithm. The algorithm was trained on our training set, then tested against the test set:

```{r}
lda.model = lda(spam ~ ., data = data.train)
lda_pred = predict(lda.model,data.test)$class
(LDA_confusion = table(lda_pred, data.test$spam))
(LDA_accuracy = mean(lda_pred == data.test$spam))
```

```
lda_pred email spam
   email   894  113
   spam     44  483
[1] 0.8976532
```

The LDA underperforms logistic regression and SVM with an accuracy of 89.77%.

Classification Tree

Before diving straight into the Random Forest method, we generated a model for Classification Trees. Classification Tree modeling and analysis is essentially when the predicted outcome can be divided by a node into multiple different classes. For example, if a predictor like char_freq_! is predicted to be found in a large subsection of total emails, those emails can then be split and more predictors can be used to further divide the data and then eventually classify them into 'Spam' or 'Email' by having met a specific criteria imposed by the model.

However, a potential issue brought forth by this model is that it can fairly easily overfit[1] the predicted outcomes and classes. The first few attempts with the CART model made it seem pretty obvious that the data was being overfit, which is why we went with the approach of growing the tree first and pruning it after to maximize the accuracy of the model. So, what is the model actually doing in terms of how selecting the best node and how does it define the classification criteria?

First for any node A, the frequencies $\widehat{p}_k$ are estimated with the following equation:

$$\widehat{p}_k = \frac{\sum_i 1\{y_i = k\}\ 1\{x_i \in A\}}{\sum_i 1\{x_i \in A\}}$$

Before the initial split, it evaluates the impurity for the entire node, A, by using the Gini Index with K different classes:

$$Gini = \sum_{k=1}^{K} \widehat{p}_k \left(1 - \widehat{p}_k\right)$$

Since we want each node to be as pure as possible, the model makes sure that the sum of their Gini impurities are as small as possible. To maximize the Gini reduction after the split:

$$score = Gini(A) - \frac{A_L}{|A|} Gini(A_L) - \frac{A_R}{|A|} Gini(A_R)$$

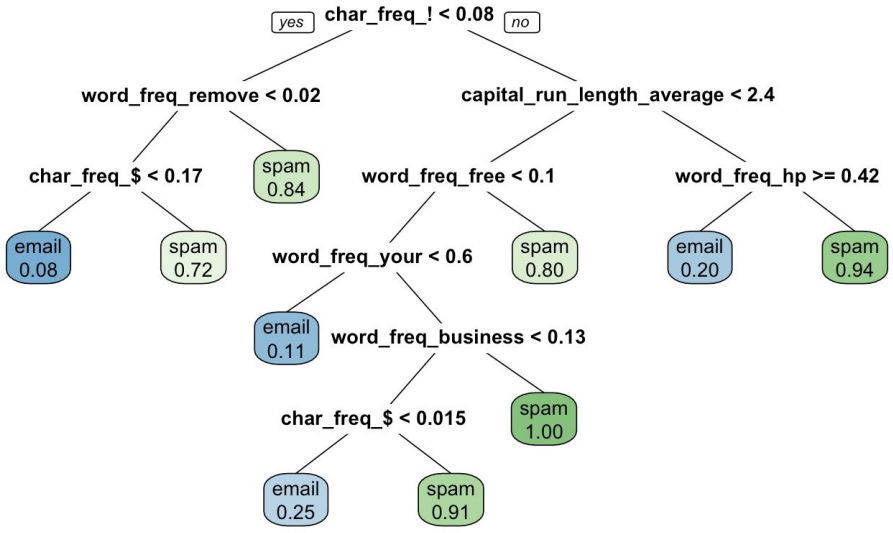
Where |•| denotes the sample size of a node and Gini(A_L) and Gini(A_R) are calculated within their respective nodes. To define a split $1\{X^{(j)} \leq c\}$ where j is the variable index and c is the cutting point, and so to find the best split the model goes through each

[1] Notes: A large tree with too many splits can overfit the data (small terminal node means small bias/large variance. The opposite is true with a small tree (large terminal node means large bias/small variance). Tree size is determined by the # of splits.

variable *j* and all of its cutting points *c*. For each combination of those two, the score of the split is calculated and compared to choose the best score.

For this spam dataset, we are using the 'rpart' package in R to generate the model for the classification tree, with the code below for reference.

```{r}
#CART model
cart <- rpart(spam ~ ., data = data.train, control = rpart.control(cp = 0.01))
bestcp <- cart$cptable[which.min(cart$cptable[,"xerror"]),"CP"]
tree.pruned <- prune(cart, cp = bestcp)

# predict classes for the evaluation data set
x.rp.pred <- predict(tree.pruned, type="class", newdata=data.test)

# score the evaluation data set (extract the probabilities)
x.rp.prob <- predict(tree.pruned, type="prob", newdata=data.test)

#CART confusion matrix
conf.matrix <- table(`Actual Class` = data.test$spam, `Predicted Class` = x.rp.pred)
Conf.matrix

#CART tree
prp(cart, extra = 104, box.palette = "auto", trace = 1)
```



As a result, the tree is pruned and generated demonstrating different splits and classification output. The pruning process was done by finding the optimal CP value which controls the tree size and minimizes the X-val relative error. In addition to that, the resultant confusion matrix showed an accuracy of 89.83% for correct classification of spam emails.

Random Forest

The Random Forest model is similar to the Classification Tree model with a couple of notable differences such as being able to perform well on high-dimensional data, utilizes variable importance (which acts like cross-validation) by running through the number of trees to select the most important variables, all while providing stability. For this model, we tuned for 1000 trees to be generated, and left mtry to the default which just ultimately takes the square-root of p, where p is equal to the number of features used in the model.

```{r}
#Random Forest training
valid_column_names <- make.names(names=names(data.train), unique=TRUE, allow_ = TRUE)
names(data.train) <- valid_column_names
pc <- proc.time()
spamRF.train = randomForest(spam~., data=data.train, ntree=1000)
proc.time() - pc

#summary statistics for RF.train
spamRF.train
rfpredvctrain <- predict(spamRF.train, newdata=data.train)

#RF test prediction
valid_column_names <- make.names(names=names(data.test), unique=TRUE, allow_ = TRUE)
names(data.test) <- valid_column_names
#summary statistics for RF.test
rfpredvct <- predict(spamRF.train, newdata=data.test)
spamRF.test  <- confusionMatrix(rfpredvct, data.test$spam)
spamRF.test
accuracy.rf <- sum(diag(spamRF.test$table))/sum(spamRF.test$table)
print(paste0("Accuracy (Precision): ", round(accuracy.rf*100, 4), paste0(" %")))
```

```
        OOB estimate of  error rate: 5.09%
Confusion matrix:
      email spam class.error
email  1791   59  0.03189189
spam     97 1120  0.07970419
Confusion Matrix and Statistics

          Reference
Prediction email spam
     email   907   35
     spam     31  561

               Accuracy : 0.957
                 95% CI : (0.9456, 0.9666)
    No Information Rate : 0.6115
    P-Value [Acc > NIR] : <2e-16

                  Sensitivity : 0.9670
                  Specificity : 0.9413
               Pos Pred Value : 0.9628
               Neg Pred Value : 0.9476
                   Prevalence : 0.6115
               Detection Rate : 0.5913
         Detection Prevalence : 0.6141
            Balanced Accuracy : 0.9541

             'Positive' Class : email

[1] "Accuracy (Precision): 95.6975 %"
```

The resultant accuracy from the Random Forest model is 95.698% which makes it look promising for choosing the ideal model for spam email classification.

Conclusion

The results show that the Random Forests model produced the most accurate classification, with SVM coming in second, the CART model in third, and KNN in last. However, while our analysis primarily focuses on getting immediate results, we can go a step further and perform cross-validation on all of the models which means we can legitimately compare the accuracy of our findings to find an optimal model for spam classification. So, the following code are models generated using the 'Caret' package along with a train-control option imposed for the cross-validation.

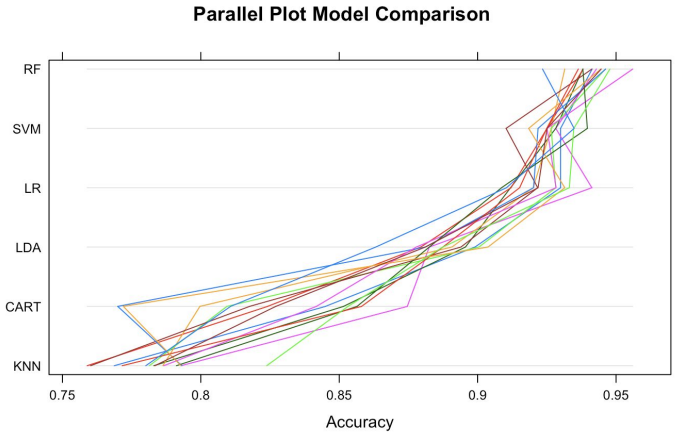
```{r}
library(caret)
control <- trainControl(method="repeatedcv", number=5, repeats=3)
#CART
cartfitted <- train(spam ~ ., data=data.train, method='rpart', trControl=control)
# LDA
set.seed(9406)
LDAfitted <- train(spam~., data=data.train, method="lda", trControl=control)
# SVM
set.seed(9406)
SVMfitted <- train(spam~., data=data.train, method="svmRadial", trControl=control)
#kNN
set.seed(9406)
KNNfitted <- train(spam~., data=data.train, method="knn", trControl=control)
#RF
set.seed(9406)
RFfitted <- train(spam~., data=data.train, method="rf", trControl=control, ntree = 1000)

LOGfitted <- train(spam~., data=data.train, method="glm", family='binomial', trControl=control)
```

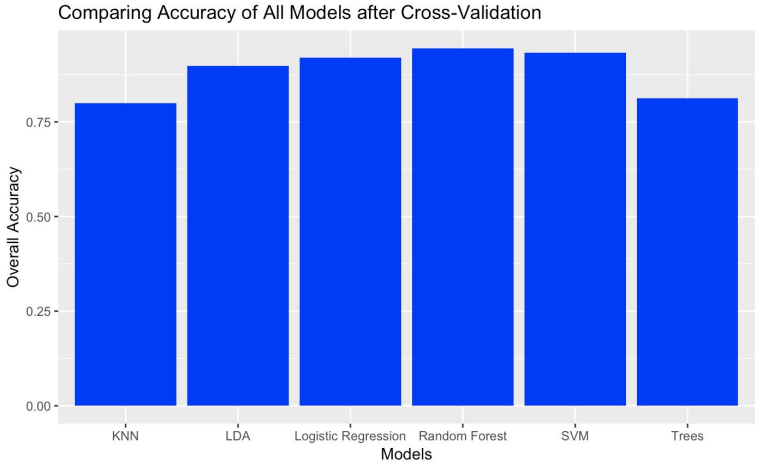
After running this chunk, we now perform analysis on the accuracy of all of the models used in our analysis. For this, we will be using three graphs showing a histogram for comparison, a box-and-whisker plot, and a parallel plot.

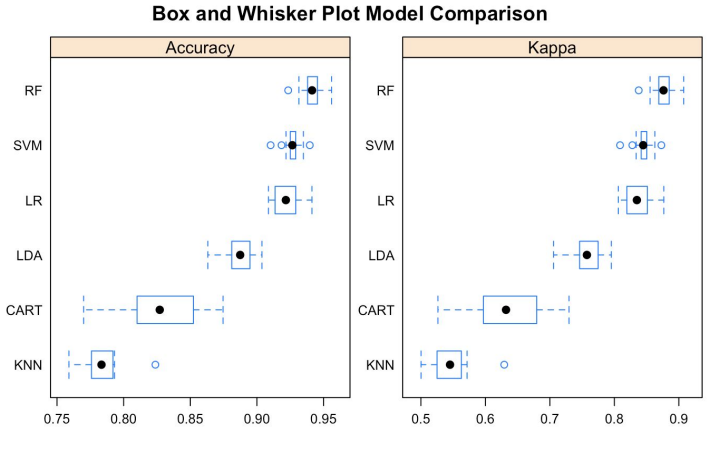
```{r}
#resamples list
ALLmodels <- resamples(list(CART=cartfitted, LDA=LDAfitted, SVM=SVMfitted,
KNN=KNNfitted, RF = RFfitted, LR = LOGfitted))

#model summary after cross-validation
summary(ALLmodels)
```

```
# box and whisker for model comparison
scales <- list(x=list(relation="free"), y=list(relation="free"))
bwplot(ALLmodels, scales=scales)

# parallel plots to compare models
parallelplot(ALLmodels)
```

```
Models: CART, LDA, SVM, KNN, RF, LR
Number of resamples: 15

Accuracy
          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
CART 0.7699837 0.8101065 0.8270799 0.8274174 0.8523654 0.8745928    0
LDA  0.8631922 0.8811075 0.8874388 0.8870819 0.8947798 0.9037520    0
SVM  0.9102773 0.9250203 0.9265905 0.9266382 0.9290946 0.9396411    0
KNN  0.7589577 0.7758727 0.7833876 0.7836138 0.7920065 0.8238173    0
RF   0.9234528 0.9380098 0.9413681 0.9410930 0.9453959 0.9560261    0
LR   0.9086460 0.9136117 0.9216966 0.9222922 0.9291531 0.9412724    0
```







To wrap everything up, these graphs demonstrate the model accuracy after doing a repeated cross-validation with 5 folds and 3 repeats, we see that the Random Forest outperforms the rest in terms of accuracy; however, since the RF model is running through 1,000 trees as well as 15 resamples, the computational time takes a lot longer than all of the other models, so factoring in efficiency SVM would be the ideal model to choose to filter out spam emails.