Crystal Mosley
MSDS 454 Sec 55

Individual Assignment #4

Introduction

The purpose of this report is to utilize the *Spam Assassin Data* to analyze and predict which emails are spam, and which are not ("ham"). During this analysis, techniques were performed like NLP (natural language processing) to either funnel out characters (i.e.: exclamation points, etc.), group stop words, uncapitalize all words so they are all the same, or to remove attachments or files that are not a part of the body of the emails. Being able to remove spam emails from your inbox helps to clear a lot of reading and the potential of opening a virus; however, sometimes emails that are not actually spam, will be removed and the user will never know about those messages unless they review their spam box. This can lead to issues on a personal and corporate level.

Data Quality Check

The data utilized for this analysis is are the *easy_ham*, *easy_ham_2*, *hard_ham*, *spam*, and *spam_2* provided by the case study from Data Science in R. The data set contains 9348 observations and 30 variables (29 predictor variables, and 1 response variable named *isSpam*) that could be derived from an email message and used for classifying spam. Of the 30 total variables, 17 are Boolean factor variables and the remaining 13 variables are numeric variables. Each email has been previously classified as spam or valid. **Table 1** displays the Variable, Type, and Definition of each.

Table 1: Variables from SpamAssassin Data

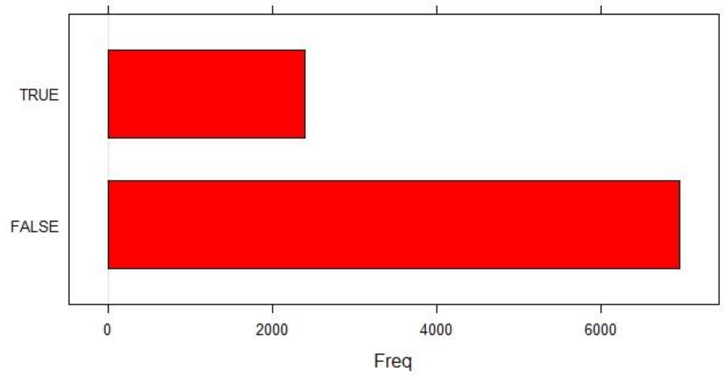
Variable	Type	Description
isSpam	logical	TRUE is email message is classified as spam.
isRe	logical	TRUE if Re: appears at the start of the subject.
numLines	integer	Number of lines in the body of the message.
bodyCharCt	integer	Number of characters in the body of the message.
underscore	logical	TRUE if email address in the From field of the header contains an underscore.
subExcCt	integer	Number of exclamation marks in the subject.
subQuesCT	integer	Number of question marks in the subject.
numATT	integer	Number of attachments in the message.
priority	logical	TRUE if a Priority key is present in the header.
numRec	numeric	Number of recipients of the message, including CCs.
perCaps	numeric	Percentage of capitals among all letters in the message body, excluding attachments.

isInReplyTo	logical	TRUE if the In-Reply-To key is present in the header.
sortedRec	logical	TRUE if the recipients' email addresses are sorted.
subPunc	logical	TRUE if words in the subject have punctuation or numbers embedded in them, e.g., w!se.
hour	numeric	Hour of the day in the Date field
multipartText	logical	TRUE if the MIME type is multipart/text.
hasImages	logical	TRUE if the message contains images.
isPGPsigned	logical	TRUE if the message contains a PGP signature.
perHTML	numeric	Percentage of characters in HTML tags in the message body in comparison to all characters.
subSpamWords	logical	TRUE if the subject contains one of the words in a spam word vector.
subBlanks	numeric	Percentage of blanks in the subject.
noHost	logical	TRUE if there is no hostname in the Message-Id key in the header.
numEnd	logical	TRUE if the emails sender's address (before the @) ends in a number.
isYelling	logical	TRUE if the subject is all capital letters.
forwards	numeric	Number of forward symbols in a line of the body, e.g., >>>xxx contains 3 forwards.
isOrigMsg	logical	TRUE if the message body contains the phrase original message.
isDear	logical	TRUE if the message body contains the word dear.
isWrote	logical	TRUE if the message contains the phrase wrote:.
avgWordLen	numeric	The average length of the words in a message.
numDlr	numeric	Number of dollar signs in the message body.

Exploratory Data Analysis

After initial data quality checks, I performed EDA on the data set. Of the total observations, 2397 were classified as spam (~26% of the full data set), and 6951 were classified as valid, as shown in **Figure 1**. The summary of the email dataframe also showed NA's for the following predictor variables: *subExcCt*, *subQuesCt*, *numRec*, *subSpamWords*, *subBlanks*, *noHost*, and *isYelling*. Rather than ridding the data set of these variables and their NA's, we imputed each variable.

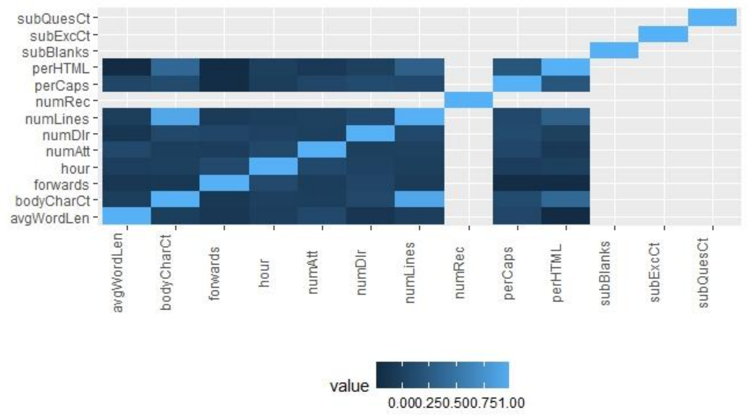
Figure 1: Spam vs. Ham Messages



Next, I created a correlation matrix to see if there exist any positive correlations between the numerical predictors. **Figure 2** shows three positive relationships:

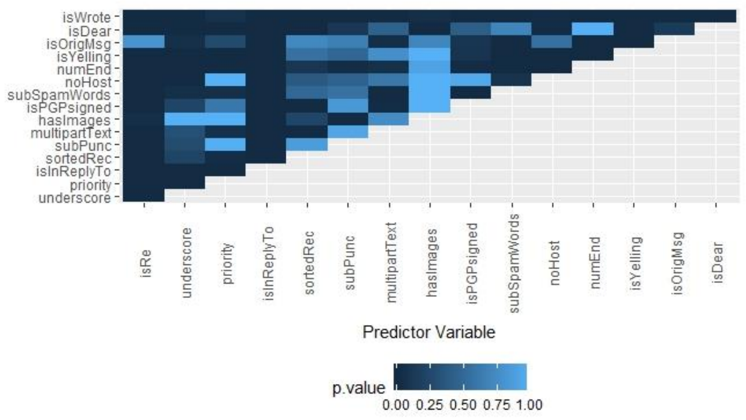
- numLines – bodyCharCt
- perHTML – bodyCharCt
- numDir – subQuesCt

Figure 2: Correlations Between Numerical Vars



Since this dataset contains 16 Boolean predictor variables, I utilized Fisher's exact test of independence to find the p-values, as shown in **Figure 3**. This test is used when there are two nominal variables and you want to see whether the proportions of one variable are different depending on the value of the other variable. The lower p-values specify that we should reject the null hypothesis of random association between these variables.

Figure 3: Fisher's Exact Test of Independence on Boolean Vars



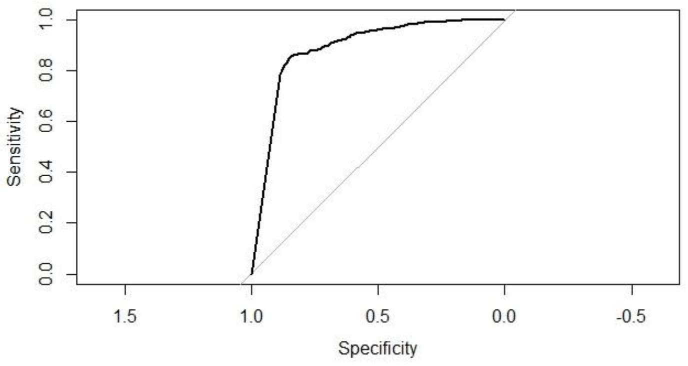
The Model Build

Now I move on to the building of my models. First, I split the data set into two sets: training and test; this way, I could use cross-validation to evaluate the accuracy of my models. The training set contained 70% of the data and the test set contained the rest of the observations. The models used were: Naïve Bayes, Decision Tree, Random Forest, Logistic Regression using Stepwise variable selection, and a Support Vector Machine. We will look at the ROC curves, AUC's, and Type I and Type II errors for each model build. The ROC (receiver operating characteristic) curve is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters: true positive rate, and false positive rate. The AUC (area under the curve) measures the entire two-dimensional area underneath the entire ROC curve from (0,0) to (1,1). The AUC provides an aggregate measure of performance across all possible classification thresholds.

Model 1: Naïve Bayes

The first model to be fit is a naïve bayes on the training data. When the assumption of independence holds, a naïve bayes classifier performs better compare to other models like logistic regression and you need less training data. It performs well in case of categorical input variables compared to numerical variables. After reviewing the results, the a-priori probability came out to about 61% of messages not being spam and about 39% of messages being spam. After reviewing both the probabilities of a-priori and conditional on each predictor variable, I created a ROC curve - **figure 4**. The curve is not as close to the top left corner which could mean that this model's ability to distinguish between spam or not spam is not as strong. This model's AUC (area under the curve) is 0.887, which is good for a first model, but could be better as we move along with the other models.

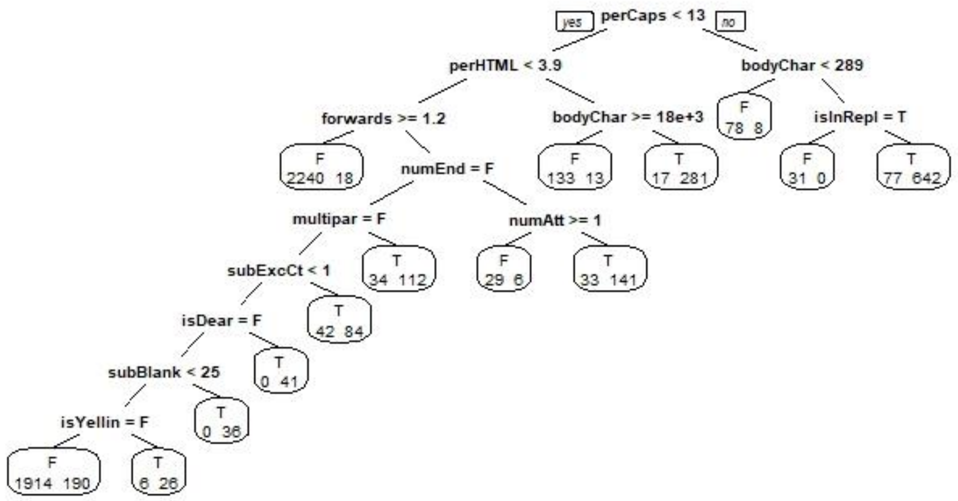
Figure 4: Naïve Bayes ROC Curve



Model 2: Decision Tree

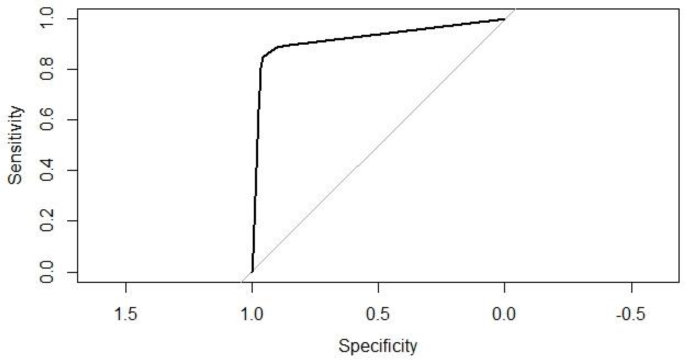
The next model to be fit is a tree model on the training data. Decision trees allow us to find the best variables for splitting while pruning unimportant variable splits. **Figure 5** shows a tree plot with 14 splits and each node specifying whether each predictor variable is T (true) or F (false) and the proportion of each class.

Figure 5: Decision Tree of predictor variables



After reviewing the tree, I created a ROC curve to measure the fit and predictive capability of the model, as show in **figure 6**. The curve is not as close to the top left corner which could mean that this model's ability to distinguish between spam or not spam is not as strong. This model's AUC (area under the curve) is 0.9223.

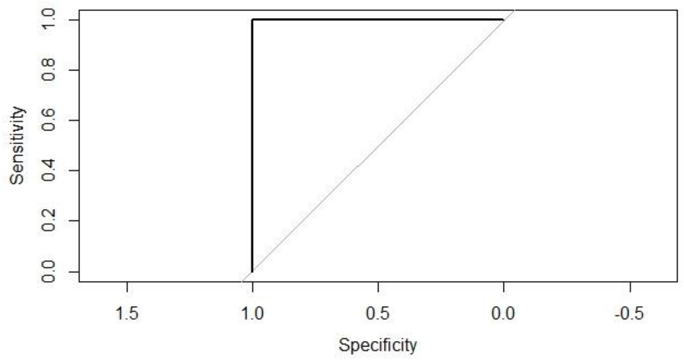
Figure 6: Decision Tree ROC Curve



Model 3: Random Forest

The third model to fit is the random forest model. I created a forest of 500 trees using 10-fold cross-validation with each split trying 29 variables. It had a 5.18% out-of-bag error rate estimate which is a method of measuring the prediction error of random forests, boosted decision trees, and other machine learning models utilizing bagging to sub-sample data samples used for training. I then created a ROC curve, **figure 7**, to measure the fit and predictive capability of the model. This model's curve is perfectly fit in the upper left corner, and it's AUC is 1 – which is also a perfect fit.

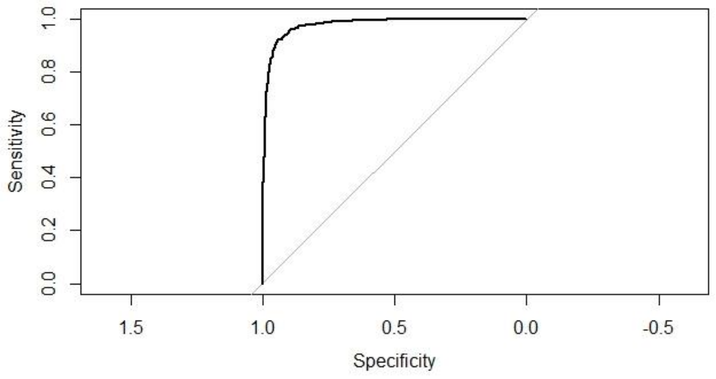
Figure 7: Random Forest ROC Curve



Model 4: Logistic Regression using Stepwise variable selection

After exploring multiple variable selection algorithms (forward, backward, etc.), I used Stepwise selection based on the lowest AIC value. After review of the coefficient estimates, I created a ROC curve to measure the fit and predictive capability of the model. **Figure 8** shows the ROC curve based on the training set of predictions and because the curve is close to the top left corner of the plot, this tells us that the model has a strong ability to distinguish between what is spam and what is not spam. This model's AUC (area under the curve) is 0.9781.

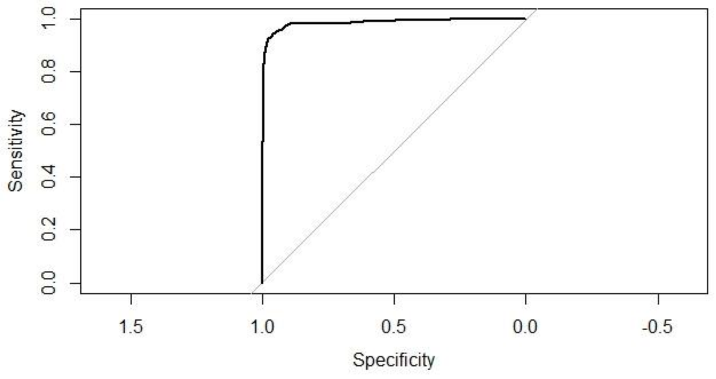
Figure 8: Stepwise ROC Curve



Model 5: Support Vector Machine

The last model fit is an SVM. I used the *svmRadial* method as well as *10-fold cross-validation* for this model. After fitting the training data, this model had a total of 1101 support vectors with an error rate of 0.043775. **Figure 9** shows the ROC curve being close to the top left corner of the plot. This tells us that the model has a strong ability to distinguish between what is spam and what is not spam. This model's AUC (area under the curve) is 0.9858.

Figure 9: SVM ROC Curve



Model Comparison

After fitting each model, I then created confusion matrices of their respective. **Table 2** shows the model type and both the in and out of sample performance for non-spam and spam. I then calculated the accuracy of each model performance for both in-sample and out-of-sample, as shown in **Table 3**.

Table 2: Confusion Matrices

Model	In-Sample			Out-of-Sample		
		Non_Spam	Spam		Non_Spam	Spam
	Non_Spam	1130	67	Non_Spam	479	29
Naïve Bayes	Spam	828	1196	Spam	351	521
		Non_Spam	Spam		Non_Spam	Spam
	Non_Spam	1873	194	Non_Spam	794	101
Decision Tree	Spam	85	1069	Spam	36	449
		Non_Spam	Spam		Non_Spam	Spam
	Non_Spam	1957	1	Non_Spam	806	59
Random Forest	Spam	1	1262	Spam	24	491
		Non_Spam	Spam		Non_Spam	Spam
Logistic Regression –	Non_Spam	1866	137	Non_Spam	801	66
Stepwise	Spam	92	1126	Spam	29	484
		Non_Spam	Spam		Non_Spam	Spam
Support Vector	Non_Spam	1901	92	Non_Spam	804	66
Machine	Spam	57	1171	Spam	26	484

Table 3: Accuracy of Model Performance

Model	In-Sample	Out-of-Sample
Naïve Bayes	0.7221	0.7246
Decision Tree	0.9134	0.9007
Random Forest	0.9994	0.9399
Logistic Regression – Stepwise	0.9289	0.9312
Support Vector Machine	0.9537	0.9333

Conclusion

After initial analysis, the Random Forest Model looks to have performed the best for the in-sample data, however, did the worst for the out-of-sample data; clearly this is an example of overfitting the data. The Naïve Bayes model did the worst of all the models as the accuracy was well below the 90%+ as shown by the other models. The Stepwise logistic model didn't perform the best, but it did perform well on the in-sample data and even better on the out-of-sample data; this is the only model with 90%+ accuracy that did better on the test data than the training. The overall model that wins for best model is the Support Vector Machine.

Appendix A: Relevant R Code

```
#----------------
#Start of EDA
#----------------

emailDF = createDerivedDF(emailStruct)
dim(emailDF)
#9348 obs, 30 variables

save(emailDF, file = "spamAssassinDerivedDF.rda")

load("C:/Users/crmo/Desktop/Class/Predict 454/Individual Ass. 4/spamAssassinDerivedDF.rda")
dim(emailDF)

names(emailDF)
str(emailDF)

summary(emailDF)
  #NA's for subExcCt, subQuesCt, numRec, subSpamWords, subBlanks, noHost, isYelling

sapply(emailDF, class)

summary(emailDF$isSpam)
2397/9348 # 0.2564 are spam

emailDFrp = setupRpart(emailDF)

# build correlation matrix
AsVector <- emailDFrp[, c(2:30)]
nums <- sapply(AsVector, is.numeric)
bools <- sapply(AsVector, is.factor)
# correlation matrix for numerical features
cormat<- (round(cor(AsVector[, nums]), 2))
cormat <- reshape::melt(cormat, na.rm=TRUE, cache=TRUE, fig.height=8, fig.width=8)
cormat %>% ggplot(aes(x=X1, y=X2, fill=value)) +
  geom_tile() +
  theme(legend.position = "bottom",
        axis.text.x = element_text(angle=90,
                                   vjust=-.5)) +
  scale_x_discrete("") +
  scale_y_discrete("")

# chi-square matrix for categorical features
Dat <- AsVector[, bools]
combos <- combn(ncol(Dat), 2)
chisqres <- adply(combos, 2, function(x) {
  test <- chisq.test(Dat[, x[1]], Dat[, x[2]])
  out <- data.frame("Row" = colnames(Dat)[x[1]]
                 , "Column" = colnames(Dat[x[2]])
                 , "Chi.Square" = round(test$statistic,3)
                 ,  "df"= test$parameter
                 ,  "p.value" = round(test$p.value, 3)
  )
  return(out)
})
chisqres %>%
  ggplot(aes(x=Row, y=Column, fill=p.value)) +
  geom_tile() +
  theme(legend.position = "bottom",
        axis.text.x = element_text(angle=90, vjust=0.5)) +
  scale_y_discrete("") + scale_x_discrete("Predictor Variable")

#----------
#Models
#----------

  #-------------------------
  #Naive Bayes model
  #-------------------------

  set.seed(123)
  nbModel <- naiveBayes(y ~., data = train)
  nbModel
    #Not_Spam      Spam
    #0.6078857 0.3921143

  #predict train
  set.seed(123)
  nbModelPred <- predict(nbModel, newdata = train, type = "raw")[,2]
  length(nbModelPred)
    #3221

  #find AUC
  set.seed(123)
  nbModelROC <- plot.roc(train$y, nbModelPred)
  nbModelAUC <- nbModelROC$auc
  nbModelAUC
    #AUC: 0.887

  #predict train for confusion matrix
  set.seed(123)
  nbModelPred2 <- predict(nbModel, newdata = train)

  set.seed(123)
  nbModelConMat <- confusionMatrix(nbModelPred2, train$y)
```

```
nbModelConMat
  #Prediction Not_Spam Spam
  #Not_Spam     1130   67
  #Spam          828   1196

#predict test
set.seed(123)
nbModelPredTest <- predict(nbModel, newdata = test)

set.seed(123)
nbModelConMatTest <- confusionMatrix(nbModelPredTest, test$y)
nbModelConMatTest
  #Prediction Not_Spam Spam
  #Not_Spam      479   29
  #Spam          351   521

#---------------
#tree model
#---------------

set.seed(123)
treeModel <- rpart(y ~ ., data = train)
treeModel

treeModelPlot <- fancyRpartPlot(treeModel, sub = "")
dev.off()

summary(treeModel)

#predict train
set.seed(123)
treeModelPred <- predict(treeModel, newdata = train, type = "prob")[,2]
length(treeModelPred)
  #3221

#find AUC
set.seed(123)
treeModelROC <- plot.roc(train$y, treeModelPred)
treeModelAUC <- treeModelROC$auc
treeModelAUC
  #AUC: 0.9223

#predict train for confusion matrix
set.seed(123)
treeModelPred2 <- predict(treeModel, newdata = train)
length(treeModelPred2)
  #6442
  #make predictions just Spam or Not_Spam
  treeModelPred2Binary <- data.frame(treeModelPred2)

  treeModelPred2Binary$y <- NA
  head(treeModelPred2Binary)
  treeModelPred2Binary[,1]

  #convert treeModelPred2Binary to factor
  treeModelPred2Binary$y <- as.factor(treeModelPred2Binary$y)
  levels(treeModelPred2Binary$y) <- c("Not_Spam", "Spam")
  summary(treeModelPred2Binary$y)

  for (i in 1:nrow(treeModelPred2Binary)){
    treeModelPred2Binary[i,3] <- ifelse(treeModelPred2Binary[i,1] > treeModelPred2Binary[i,2],
"Not_Spam", "Spam")
  }
  head(treeModelPred2Binary)
  summary(treeModelPred2Binary$y)
    #Not_Spam     Spam
    #2067     1154

  set.seed(123)
  treeModelConMat <- confusionMatrix(treeModelPred2Binary$y, train$y)
  treeModelConMat
    #Prediction Not_Spam Spam
    #Not_Spam     1873  194
    #Spam           85 1069

  #predict test
  set.seed(123)
  treeModelPredTest <- predict(treeModel, newdata = test)
  length(treeModelPredTest)
    #2760

  #make predictions just Spam or Not_Spam
  treeModelPredTestBinary <- data.frame(treeModelPredTest)
  treeModelPredTestBinary$y <- NA
  head(treeModelPredTestBinary)

  #treeModelPredTestBinary to factor
  treeModelPredTestBinary$y <- as.factor(treeModelPredTestBinary$y)
  levels(treeModelPredTestBinary$y) <- c("Not_Spam", "Spam")
  summary(treeModelPredTestBinary$y)

  for (i in 1:nrow(treeModelPredTestBinary)){
    treeModelPredTestBinary[i,3] <- ifelse(treeModelPredTestBinary[i,1] > treeModelPredTestBinary[i,2],
"Not_Spam", "Spam")
  }
  head(treeModelPredTestBinary)
  summary(treeModelPredTestBinary$y)
    #Not_Spam Spam
    #895     485

  set.seed(123)
  treeModelConMatTest <- confusionMatrix(treeModelPredTestBinary$y, test$y)
  treeModelConMatTest
    #Prediction Not_Spam Spam
    #Not_Spam      794  101
    #Spam           36  449

  #-----------------------------
#RandomForest model
#-----------------------------

set.seed(123)
rfControl <- trainControl(method = "cv", classProbs = T, savePred = T, verboseIter = T)

set.seed(123)
rfModel <- train(y ~ ., data = train, method="rf", metric="Accuracy", trControl=rfControl)
print(rfModel)

plot(rfModel)
rfModel$finalModel
  #        Not_Spam Spam class.error
  #Not_Spam     1891   67  0.03421859
  #Spam         100  1163  0.07917656

#fit rfModel
set.seed(123)
rfModelFit <- randomForest(y ~ ., data = train, mtry=29, importance=TRUE)
rfModelFit
  #         Not_Spam Spam  class.error
  #Not_Spam     1893   65    0.03319714
  #Spam           96   1167  0.07600950

#predict train
set.seed(123)
rfModelPred <- predict(rfModel, newdata = train, type = "prob")[,2]
length(rfModelPred)
  #3221

#find AUC
set.seed(123)
rfModelROC <- plot.roc(train$y, rfModelPred)
rfModelAUC <- rfModelROC$auc
rfModelAUC
```

```
  #AUC: 1

 #predict train for confusion matrix
 set.seed(123)
 rfModelPred2 <- predict(rfModel, newdata = train)

 set.seed(123)
 rfModelConMat <- confusionMatrix(rfModelPred2, train$y)
 rfModelConMat
  #Prediction Not_Spam Spam
  #Not_Spam     1957    1
  #Spam            1   1262

 #predict test
 set.seed(123)
 rfModelPredTest <- predict(rfModel, newdata = test)

 set.seed(123)
 rfModelConMatTest <- confusionMatrix(rfModelPredTest, test$y)
 rfModelConMatTest
  #Prediction Not_Spam Spam
  #Not_Spam      806   59
  #Spam           24   491

 #--------------------------------------------------------------------------
 #logistic regression model using stepwise variable selection
 #--------------------------------------------------------------------------

 # # Stepwise train
 # set.seed(123)
 # stepwise <- regsubsets(y ~ ., data = train, nvmax=NULL, method="seqrep")
 # options(max.print=1000000)
 # summary(stepwise)
 #
 # # Model using top 15 predictors from Stepwise
 # set.seed(123)
 # stepwiseFit <- train(y ~ word_freq_your + word_freq_000 + word_freq_remove + word_freq_free +
capital_run_length_total + word_freq_money + char_freq_exclamation + word_freq_our +
word_freq_hp + char_freq_usd,  data=train, method="glm", family="binomial")
 # stepwiseFit$finalModel

 logitControl <- trainControl(classProbs = T, savePred = T , verboseIter = T)
 set.seed(123)
 stepwise2 <- train(y ~ ., data = train, method = "glmStepAIC",
                 direction = "forward", trControl = logitControl)

 names(stepwise2)
 stepwise2$finalModel
```

```
summary(stepwise2$finalModel)
#AIC = 1330.207

length(stepwise2$finalModel$coefficients)-1
#find number of predictors used: 37
AIC(stepwise2$finalModel)
#1330.207
stepwise2$metric
#Accuracy

#predict train
set.seed(123)
stepwise2Pred <- predict(stepwise2, newdata = train, type = "prob")[,2]
length(stepwise2Pred)
#3221

#find AUC
set.seed(123)
stepwise2ROC <- plot.roc(train$y, stepwise2Pred)
stepwise2AUC <- stepwise2ROC$auc
stepwise2AUC
#AUC: 0.9781

#predict train for confusion matrix
set.seed(123)
stepwise2Pred2 <- predict(stepwise2, newdata = train)

set.seed(123)
stepwise2ConMat <- confusionMatrix(stepwise2Pred2, train$y)
stepwise2ConMat
#Prediction Not_Spam Spam
#Not_Spam     1866  137
#Spam           92 1126

#predict test for confusion matrix
set.seed(123)
stepwise2Pred2Test <- predict(stepwise2, newdata = test)

set.seed(123)
stepwise2ConMatTest <- confusionMatrix(stepwise2Pred2Test, test$y)
stepwise2ConMatTest
#Prediction Not_Spam Spam
#Not_Spam      801   66
#Spam           29  484

#---------------
#SVM model
#---------------
```

```
svmControl <- trainControl(method = "cv", classProbs = T, savePred = T, verboseIter = T)

set.seed(123)
svmModel <- train(y ~ ., data = train, method = "svmRadial", trControl = svmControl)

svmModel$finalModel

#predict train
set.seed(123)
svmModelPred <- predict(svmModel, newdata = train, type = "prob")[,2]
length(svmModelPred)
#3221

#find AUC
set.seed(123)
svmModelROC <- plot.roc(train$y, svmModelPred)
svmModelAUC <- svmModelROC$auc
svmModelAUC
#AUC: 0.9858

#predict train for confusion matrix
set.seed(123)
svmModelPred2 <- predict(svmModel, newdata = train)

set.seed(123)
svmModelConMat <- confusionMatrix(svmModelPred2, train$y)
svmModelConMat
#Prediction Not_Spam Spam
#Not_Spam     1901   92
#Spam           57   1171

#predict test
set.seed(123)
svmModelPredTest <- predict(svmModel, newdata = test)

set.seed(123)
svmModelConMatTest <- confusionMatrix(svmModelPredTest, test$y)
svmModelConMatTest
#Prediction Not_Spam Spam
#Not_Spam      804   66
#Spam           26   484
```