# Spam Email Classification

By Group BEAR
Ya-Hsuan Chuo (yc3238) ,
Barbara Lu (xl2627),
Sui Ping Suen (ss5202),
Helen Tao (jt2956)

## Catalogue

Introduction………………………………………………………………………

Data cleaning and exploration…………………………………………………........

- Data summary……………………………………………………………………..
- Data cleaning……………………………………………………………………....

Naive Bayes Classifier……………………………………………………………

- Model Introduction………………………………………………………………....
- Strengths and Weaknesses………………………………………………………….
- Metrics…………………………………………………………………………………...
- Impact of training data size on test accuracy…………………………………………

Support Vector Machine (SVM) Classifier………………………………………..

- Model Introduction………………………………………………………………....
- Strengths and Weaknesses………………………………………………………….
- Metrics…………………………………………………………………………………...
- Impact of training data size on test accuracy…………………………………………

Conclusion…………………………………………………………....……………..

Bonus……………………………………………………………....……………………

- Decision Tree……………………………………………………………....…….
  - Model Introduction……………………………………………………………
  - Metrics……………………………………………………………………………...

## Introduction

The need for reliable anti-spam filters has been drastically growing because of the volume of unsolicited, undesired, and illegal email messages in this digital era. Machine learning techniques are frequently employed to build classifying models which could automatically filter those spam e-mails. In this project, we developed three spam classifiers by using Naive Bayes, Support Vector Machine (SVM), and Decision Tree techniques. Our goal of this project is to build a binary spam classifying model with best performance, which shows a steady high prediction accuracy and low False Positive Rate. We will discuss the strengths and weaknesses of each model, compute different metrics, tune the models, explore the impact of training data size on test accuracy and compare their performances. We will start our project by cleaning and exploring the dataset we got.

## Data cleaning and exploration

### Data Summary

- ❖ Spam Data comes from UC Irvine dataset repository (URL: https://archive.ics.uci.edu/ml/datasets/Spambase)
- ❖ 4601 Observations
- ❖ 57 Predictor Variables
  - ➢ 48 continuous real [0,100] attributes of type word_freq_WORD
    = percentage of words in the e-mail that match WORD
  - ➢ 6 continuous real [0,100] attributes of type char_freq_CHAR
    = percentage of characters in the e-mail that match CHAR
  - ➢ 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 e-mail
- ❖ Variable of Interest: Spam
  - ➢ 1 nominal {0,1} class attribute of type spam
    = denotes whether the e-mail was considered spam (1) or not (0)

### Data Cleaning and Exploration

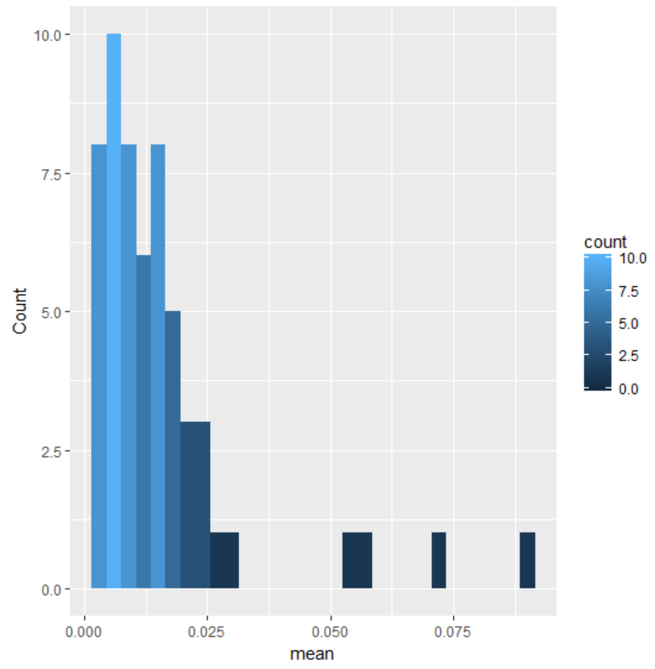
Firstly, we import the dataset into R studio with the following codes:

```
spam <- read.csv("C:/Users/Amy Chuo/Downloads/data.csv",header=FALSE,sep=";")
```

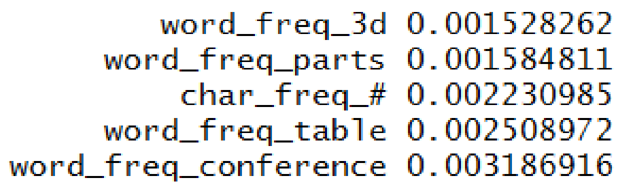
```
names <- read.csv("C:/Users/Amy Chuo/Downloads/names.csv",header=FALSE,sep=";")
names(spam) <- sapply((1:nrow(names)),function(i) toString(names[i,1]))
spam$y <- as.factor(spam$y)
```

Secondly, we load raw spam data into R, and rename the variables. We find that there is no missing value in the data with code `sum(!complete.cases(spam))`

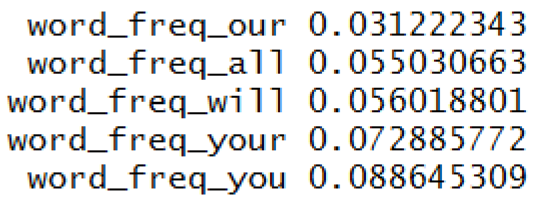
## NORMALIZATION



Most of our variables are labeled as “percentage”, which calculate the possibility of certain characters that appears in each e-mail. However, there are three variables that are continuous integers which measure the length of uninterrupted sequences of capital letters. Therefore, the large variety of units lead us to normalize this data. After normalization, the data are all range in [0,1]. This graph illustrates the mean of 57 variables and how they distribute. It shows that most of the variables are around 0 to 0.0025.



```
        word_freq_3d 0.001528262
     word_freq_parts 0.001584811
         char_freq_# 0.002230985
     word_freq_table 0.002508972
word_freq_conference 0.003186916
```



```
 word_freq_our 0.031222343
 word_freq_all 0.055030663
word_freq_will 0.056018801
word_freq_your 0.072885772
 word_freq_you 0.088645309
```

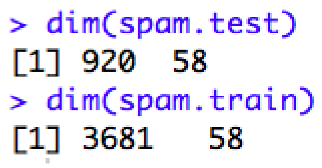
The result shows that which five variables have high possibility and five variables that have low possibility that appears in the e-mail.

### Split data

Next, we use 80:20 randomly sampled subset for training and testing datasets:

```
indexes <- sample(1:nrow(spam), size=0.2*nrow(spam))
```

```
spam.test <- spam[indexes,]
spam.train <- spam[-indexes,]
```



```
> dim(spam.test)
[1] 920  58
> dim(spam.train)
[1] 3681   58
```

## Naïve Bayes Classifier

### Model Introduction

In machine learning, Naïve Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naïve) independence assumptions between the features.

In the case of spam filtering, the response variable Y has two classes, Y=1 when the email is spam, and Y=0 when the email is not spam. The input variables can be represented by a vector, $X = (x_1,\dots,x_{57})$. X describes word frequencies, character occurrences, and capital letter lengths features in a specific email. The purpose of our model is to predict the probability of Y=1 given a specific X, which can be represented by the conditional probability $p(Y=1|x_1,\dots,x_{57})$.

According to Bayes' theorem,

$p(Y=1|x_1,\dots,x_{57}) = \frac{p(x_1,\dots,x_{57}|Y=1)\times p(Y=1)}{p(x_1,\dots,x_{57})} \quad p(x_1,\dots,x_{57})\neq 0$

The numerator is equivalent to the joint probability model

$p(Y=1,x_1,\dots,x_{57})$

which can be rewritten as follows, using the chain rule for repeated applications of the definition of conditional probability:

$p(Y=1,x_1,\dots,x_{57})$

$= p(x_1,\dots,x_{57},Y=1) = p(x_1|x_{2},\dots,x_{57},Y=1)\, p(x_2,\dots,x_{57},Y=1)$

$= p(x_1|x_{2},\dots,x_{57},Y=1)p(x_2|x_{3},\dots,x_{57},Y=1)p(x_{3},\dots,x_{57},Y=1)$

$=\dots$

$= p(x_1|x_{2},\dots,x_{57},Y=1)p(x_2|x_{3},\dots,x_{57},Y=1)\dots p(x_{56}|x_{57},Y=1)p(x_{57}|Y=1)p(Y=1)$

Now the "naive" conditional independence assumptions come into play: assume that each feature $F_i$ is conditionally independent of every other feature $F_j$ for $i\neq j$, given the category C. This means that

$p(x_i|x_{i+1},\dots,x_{57},Y=1)=p(x_i|Y=1)$.

Thus, $p(Y=1,x_1,\dots,x_{57})=p(Y=1)\prod_{i}^{57} p(x_i|Y=1)$ .

So $p(Y=1|x_1,\dots,x_{57}) = \frac{p(Y=1)\prod_{i}^{57} p(x_i|Y=1)}{p(x_1,\dots,x_{57})}$

$p(x_1,\dots,x_{57})\neq 0$

With the maximum a posteriori (MAP) decision rule, when $p(Y=1|x_1,\dots,x_{57})>0.5$, the model would predict that $\hat{y}=1$.

During this process, we can notice that Naive Bayes Classifier works only with categorical predictors. Numerical predictors must be categorized or binned before use.

We use probability density function (pdf) values, where we preserve the continuous values as such. We start by assuming the probability distribution for an attribute follows a Gaussian distribution. If it is known to follow some other distribution, such as Poisson's, the equivalent probability density function can be used.

Using the attributes' mean and standard deviation for each of the two class outcomes (Y=1 and Y=0), we estimate the pdf value at any given decision point. The two pdf values will then be used in the Naive Bayes formula to obtain the corresponding probabilities (of Y=0 and 1).

In our model, we use two parameters: threshold and eps. Threshold is a value that will replace any probabilities less than whatever we set for eps. Eps is the maximum value we would like to keep "as is". These two parameters  can impact the model accuracy and need to be tuned to get the optimal value.

**Laplace Smoothing**

If we receive an email that contains a word that has never appeared in the training emails,

p(=1|Y) will be 0 for all Y values.

We can only make prediction based on p(Y).

This is bad because we ignored all the other words in the email because of this single rare word. To avoid this situation, we use Laplace smoothing.

p(=1|Y=n) = (1+ # of examples with Y=n, =1) /(k+ # of examples with Y=n)

k= the total number of possible values of

## Strengths and Weaknesses

Strengths

1. Because independent variables are assumed, each variable distribution can be independently estimated as a one-dimensional distribution. This greatly simplifies the algorithm and helps alleviate problems stemming from the curse of dimensionality, such as the need for data sets that scale exponentially with the number of features.
2. Naïve Bayes is wonderfully cheap, fast and survives tens of thousands of attributes easily.
3. Naïve Bayes classifier performs class predictions well.

### Weaknesses

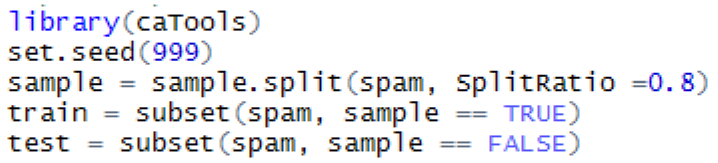
1. The strong independent assumptions are often inaccurate. Thus, Naive Bayes does not produce a good estimate for the correct class probabilities.
2. Naïve Bayes can not learn interactions between features.

## Metrics

### Initial  Model

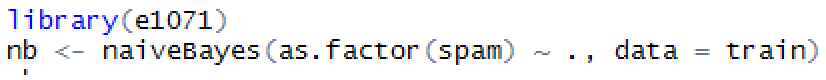
To analyze our spam database by applying Naïve Bayes, we process the following steps:

•First, we randomly split our dataset into training data and test data in the ratio of 80:20:



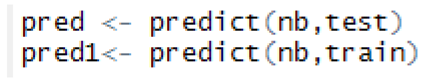
```
library(caTools)
set.seed(999)
sample = sample.split(spam, SplitRatio =0.8)
train = subset(spam, sample == TRUE)
test = subset(spam, sample == FALSE)
```

•Second, we use our training data to generate a Naïve Bayes classifier:



```
library(e1071)
nb <- naiveBayes(as.factor(spam) ~ ., data = train)
```

•Finally, we apply the model we got to predict the number of spams in both our training and test dataset:



```
pred <- predict(nb,test)
pred1<- predict(nb,train)
```

### Metrics for Initial  Model

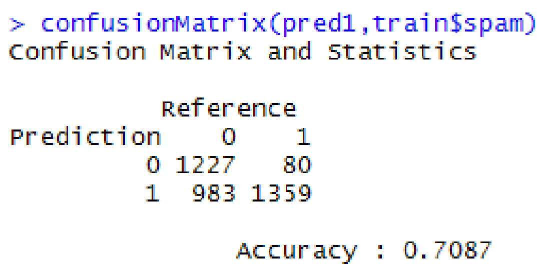
To measure the accuracy of our Naïve Bayes model, we have made the following three metrics:

1.Confusion Matrix

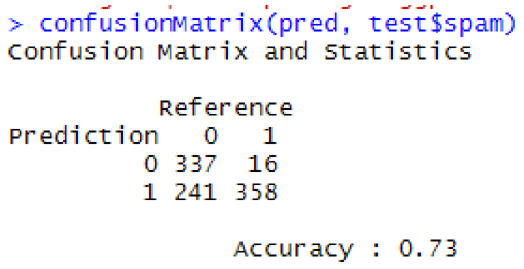
From the following results, we can see that:

For training dataset, TN=1227, FN=80, FP= 983 and TP=1359

For the test dataset, TN=337, FN=16, FP=241, TP=358



```
> confusionMatrix(pred1,train$spam)
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1227   80
         1  983 1359

               Accuracy : 0.7087
```



```
> confusionMatrix(pred, test$spam)
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 337  16
         1 241 358

               Accuracy : 0.73
```

2.Classification Accuracy

We calculate our classification accuracy by using the following formulas:

$$\text{Error}=\frac{□□+□□}{□□+□□+□□+□□}$$

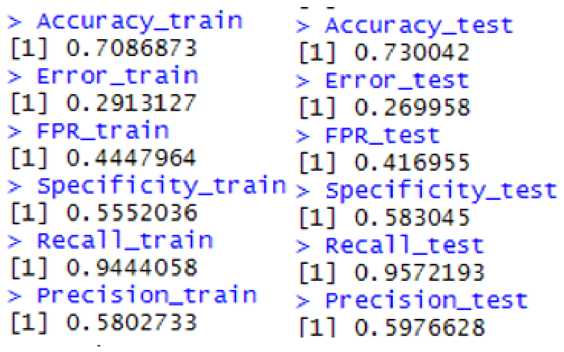
$$\text{Accuracy=1-Error}=\frac{□□+□□}{□□+□□+□□+□□}$$

$$\text{FPR(False Positive Rate)}=\frac{□□}{□□+□□} \quad \text{Specificity=1-FPR}$$

$$\text{TPR(True Positive Rate)}=\frac{□□}{□□+□□}=\text{Recall = Sensitivity}$$

$$\text{Precision}=\frac{□□}{□□+□□}$$

Here are the classification accuracies for both training and test data:

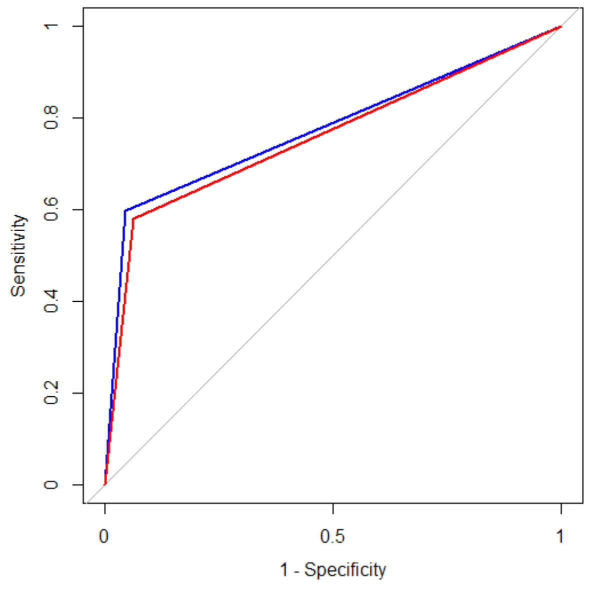


```
> Accuracy_train              > Accuracy_test
[1] 0.7086873                 [1] 0.730042
> Error_train                 > Error_test
[1] 0.2913127                 [1] 0.269958
> FPR_train                   > FPR_test
[1] 0.4447964                 [1] 0.416955
> Specificity_train           > Specificity_test
[1] 0.5552036                 [1] 0.583045
> Recall_train                > Recall_test
[1] 0.9444058                 [1] 0.9572193
> Precision_train             > Precision_test
[1] 0.5802733                 [1] 0.5976628
```

For the spam classifier, we mainly focus on the accuracy and the FPR since we do not want to make our users miss important emails. From the results, we can see that our model has a higher accuracy, but lower FPR for test dataset compared with train dataset. Meanwhile, the value of accuracy indicators of the two datasets are very close, which means that our Naïve Bayes model works well.

3. Area Under ROC Curve

To visualize the total performance of our model, we draw the ROC graph and generate the area under ROC curve(AUC). The higher the value of AUC, the better the model. In the ROC graph below, blue line represents the test data, and the red line represents the training dataset. From both the ROC graph and AUC values, we can see that our model works better for the test dataset, and both ROC lines and AUC values are very close.



```
> ROC1

Call:
roc.default(response = pred1, predictor = train$spam)

Data: train$spam in 1307 controls (pred1 0) < 2342 cases (pred1 1).
Area under the curve: 0.7595
> ROC

Call:
roc.default(response = pred, predictor = test$spam)

Data: test$spam in 353 controls (pred 0) < 599 cases (pred 1).
Area under the curve: 0.7762
```

Among these different metrics above, we believe that ROC and AUC are more relevant tools for completing our task, since ROC graph can visualize the model performance on both training and test dataset in one graph, and AUC can give exact number of how model works on different datasets for us to compare.

## Tune Model

### Holdout Cross Validation

To achieve better generalization and performance of our classifier, we need to tune our model by applying holdout cross validation. As we mentioned in the model introduction, we use two parameters, threshold and eps, in our model. In the initial model, we use default values for these two parameters. However, in the validation process, we will manually adjust and optimize the values of threshold and eps.

To accomplish this model tuning process, we implement the following steps:

•First, we further randomly split our training dataset into new training data and validation data in the ratio of 80:20

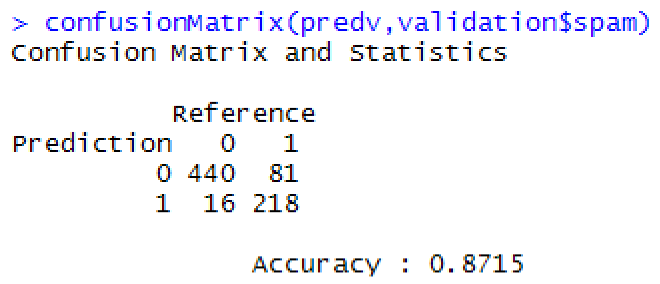
```
set.seed(999)
vali=sample.split(train,SplitRatio=0.8)
validation= subset(train,vali== FALSE)
train = subset(train,vali== TRUE)
```

•Second, we use the new training data to general the model

```
library(e1071)
nb <- naiveBayes(as.factor(spam) ~ ., data = train)
```

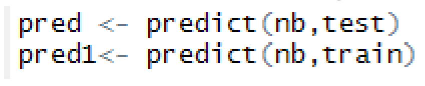
•Third, we adjust the value of threshold and eps for several times and compare the confusion matrix and accuracy each time until we find optimized values for these two parameters. The goal we want to achieve here is to keep higher accuracy and  relatively lower FPR at the same time. Here are the optimized parameters we decide to use

```
predv<- predict(nb,validation,threshold=0.000025, eps=0.018)
table(predv,validation$spam)
confusionMatrix(predv,validation$spam)
```



```
> confusionMatrix(predv,validation$spam)
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 440  81
         1  16 218

               Accuracy : 0.8715
```

•Finally, we use the optimized parameters we found in the above step to predict the number of spams. To ensure that we do not have the problem of overfitting or underfitting, we still perform our model on both training and test data



```
pred <- predict(nb,test)
pred1<- predict(nb,train)
```

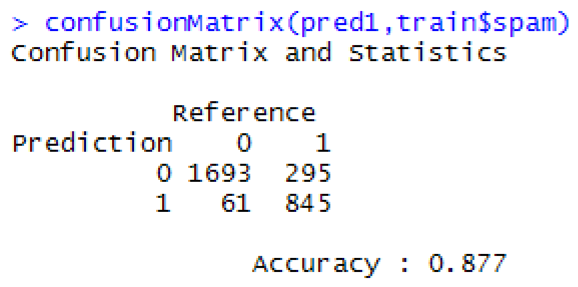
## Metrics for Tuned Model

1.Confusion Matrix

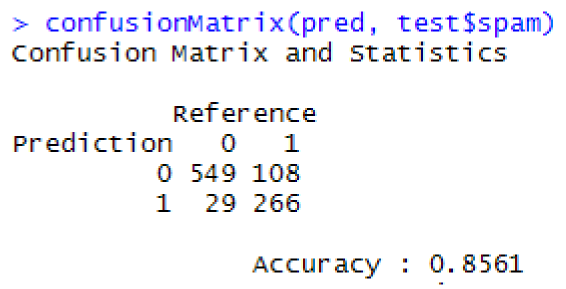
From the following results, we can see that:

For training dataset, TN=1693, FN=295, FP= 61 and TP=845

For the test dataset, TN=549, FN=108, FP=29, TP=266



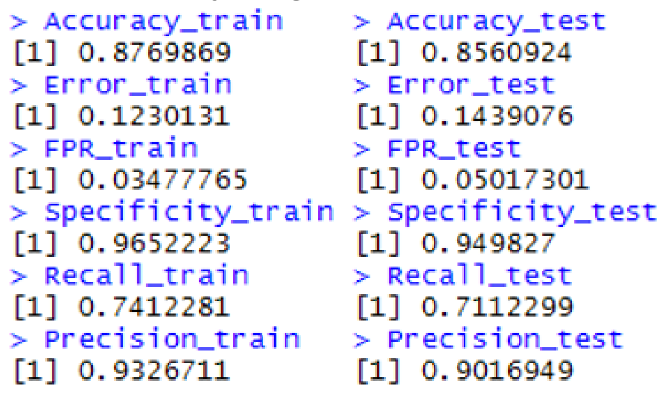
```
> confusionMatrix(pred1,train$spam)
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1693  295
         1   61  845

               Accuracy : 0.877
```



```
> confusionMatrix(pred, test$spam)
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 549 108
         1  29 266

               Accuracy : 0.8561
```

2.Classification Accuracy

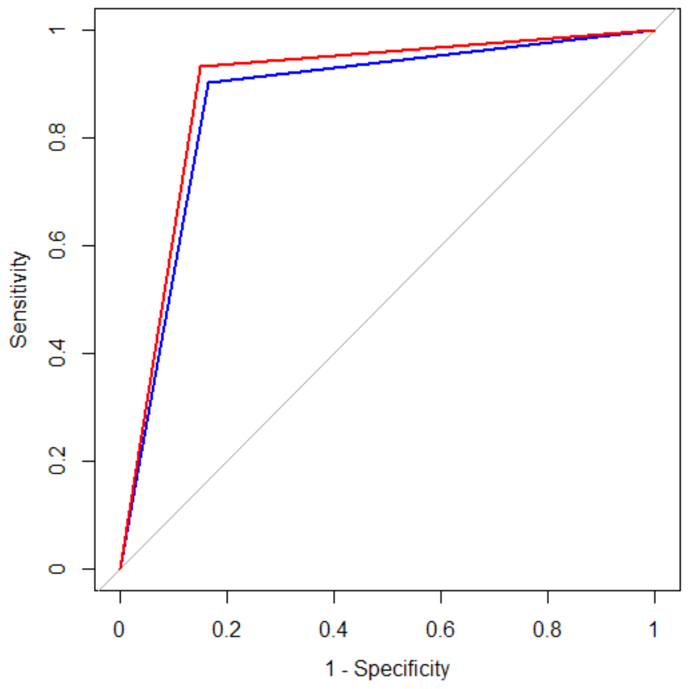
We calculate our classification accuracy using same formulas:



```
> Accuracy_train      > Accuracy_test
[1] 0.8769869         [1] 0.8560924
> Error_train         > Error_test
[1] 0.1230131         [1] 0.1439076
> FPR_train           > FPR_test
[1] 0.03477765        [1] 0.05017301
> Specificity_train   > Specificity_test
[1] 0.9652223         [1] 0.949827
> Recall_train        > Recall_test
[1] 0.7412281         [1] 0.7112299
> Precision_train     > Precision_test
[1] 0.9326711         [1] 0.9016949
```

From the results, we can see that our model has a higher accuracy, but lower FPR for training dataset. Meanwhile, the value of different accuracy indicators of the two datasets are very close, which means that our Naïve Bayes model works well. Compare to our initial model, both accuracy and FPR have been improved a lot.

3. Area Under ROC Curve

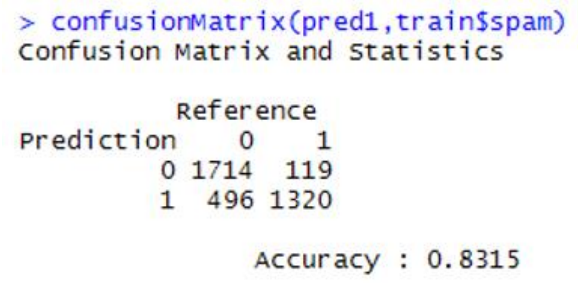


```
> ROC1

Call:
roc.default(response = pred1, predictor = train$spam)

Data: train$spam in 1988 controls (pred1 0) < 906 cases (pred1 1).
Area under the curve: 0.8921
> ROC

Call:
roc.default(response = pred, predictor = test$spam)

Data: test$spam in 657 controls (pred 0) < 295 cases (pred 1).
Area under the curve: 0.8687
```

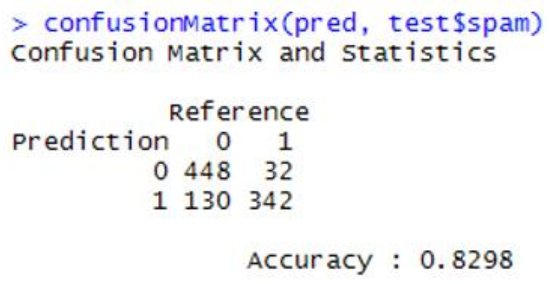
From both the ROC graph and AUC values, we can see that our model works better for the training dataset, and both ROC lines and AUC values are still very close. In general, our model has been improved a lot.

## Impact of training data size on test accuracy

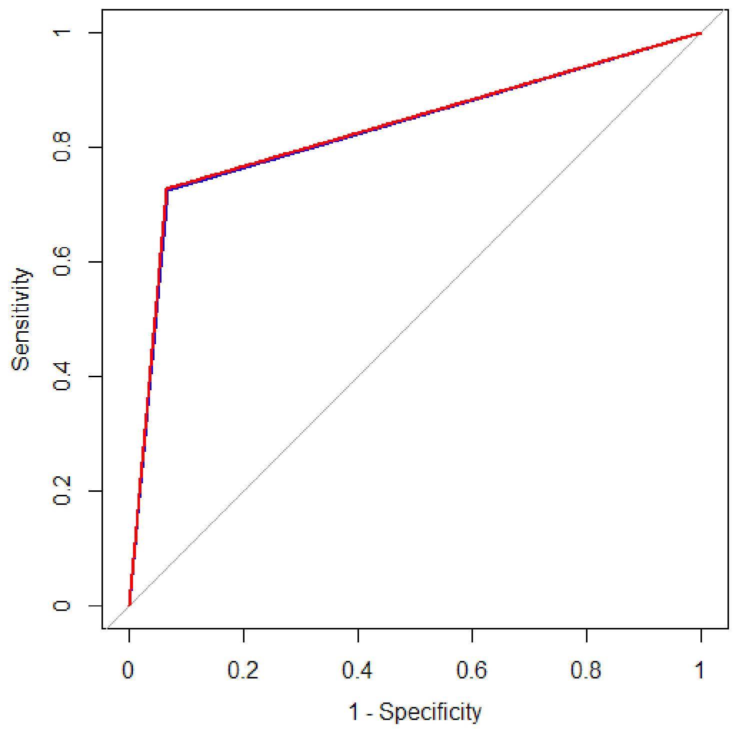
In this section, we want to discover the relationship between the size of training dataset and model accuracy. To initiate this experiment, we divide the spam database into training data and test data in the ratio of 80:20. Then we generate the Naïve Bayes model using the optimized parameters we found in the above sections, which are threshold=0.000025 and eps=0.018. We use this initial model as a control group. As a result, we got following test accuracy and ROC curve:



```
> confusionMatrix(pred1,train$spam)
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1714  119
         1  496 1320

               Accuracy : 0.8315
```



```
> confusionMatrix(pred, test$spam)
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 448  32
         1 130 342

               Accuracy : 0.8298
```

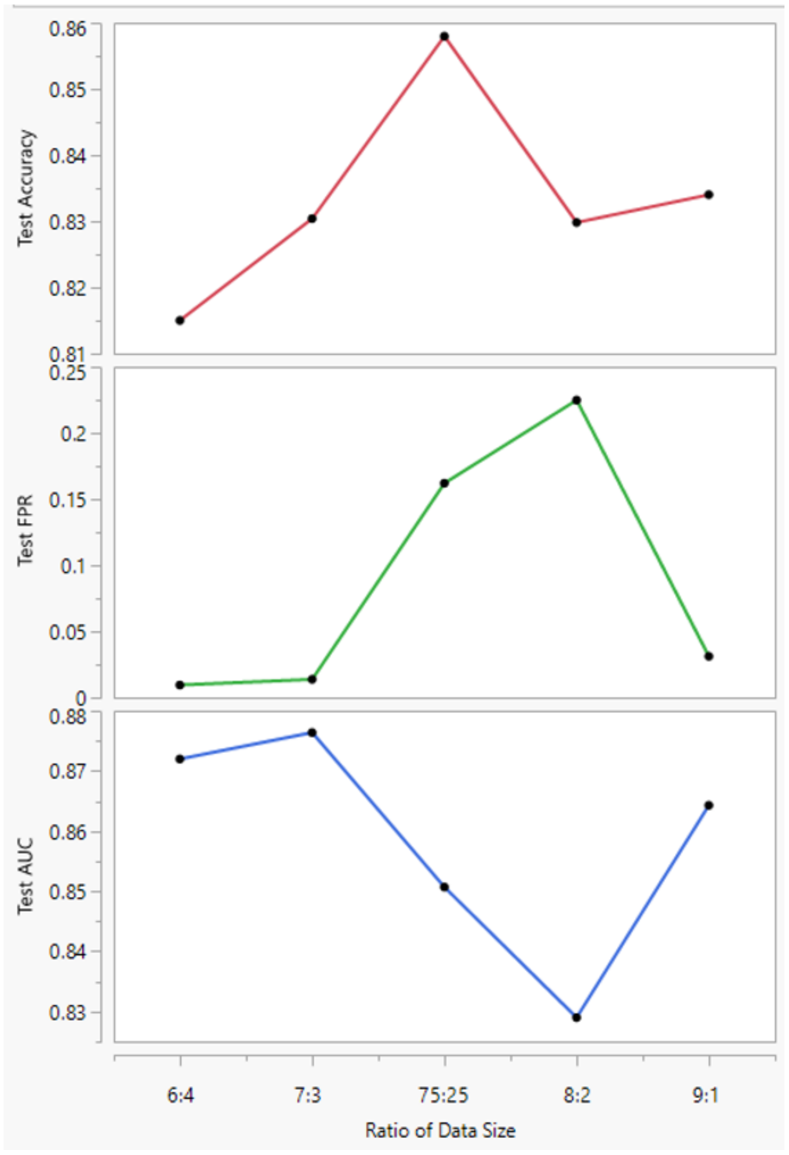


We can see that when training dataset is 80% and test dataset is 20%, the test accuracy is 82.98%, and very close to the training accuracy. Next, we try to divide the spam database into training data and test data in the ratio of 60:40, 70:30, 75:25, 90:10 respectively and generate the test accuracy and ROC graph again. The results are collected in the following table:

| Train:Test Split | 60:40 | | 70:30 | | 75:25 | | 80:20 | | 90:10 | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Train | Test | Train | Test | Train | Test | Train | Test | Train | Test |
| **Accuracy** | 0.82098 | 0.81503 | 0.83711 | 0.83041 | 0.87335 | 0.85798 | 0.83146 | 0.82983 | 0.85139 | 0.83403 |
| **FPR** | 0.01102 | 0.00953 | 0.01093 | 0.01386 | 0.13746 | 0.16205 | 0.22443 | 0.22491 | 0.01800 | 0.03125 |
| **AUC** | 0.8737 | 0.872 | 0.883 | 0.8764 | 0.8658 | 0.8507 | 0.831 | 0.829 | 0.8856 | 0.8643 |

In the table above, the highlight parts represents the one which has the lowest FPR and the highest test and training accuracies and AUC. From these results, we can see that when the training data and test data are splited in the ratio of 75:25, the test accuracy is significantly high, test AUC is on average, but the FPR of test data is relatively high. When the training and test datasets are splited in the ration of 7:3, the test accuracy is on average, FPR of test is relatively low, and the AUC of test is significantly high. In general, when we divide our training data and test data in the ratio of 7:3, the predictions on test data performs best.

We also draw a line chart to show the trend of test accuracy, FPR and AUC when the size of training dataset varies. The chart shows as follows:



From the chart above, we can see that looking from test accuracy's perspective, treat 75:25 as a central point, the model performs worse either the size of training data increases or decreases. Looking from the test FPR and AUC's perspective, the model performs worse as the size of training data increases, and the performance well again when the training data and test data are in the ratio of 9:1.

From this experiment, we have learnt that except for tuning our model parameters, the way we divide our original data into training and test datasets will also have large impact on our model performance. According to our experiment, in general, using training and test datasets with the size ratio of 7:3 makes our model perform best.

## Support Vector Machine (SVM) Classifier

### Model Introduction

SVM is a linear classifier (supervised learning) that outputs an optimal hyperplane for classification for a linearly separable dataset. Classification tasks based on drawing separating lines to distinguish between objects of different class memberships are known as hyperplane classifiers. Support Vector Machines are particularly suited to handle such tasks.

In choosing the better model of SVM, we look at different parameters. For the C parameter, a lower C makes the decision surface smooth. While a high C aims at classifying all training examples correctly thus would result in a overfitting problem. Second, for the kernel function, we tried for the linear, radial, polynomial. These are viewed as the most commonly used kernel.

### Strengths and Weaknesses

#### Strengths

1. The SVM model gives us a unique solution, since the optimality problem is convex. In Mar. 10th class we know that SVM is a convex quadratic programming (QP) problem, so the global optimum solution is guaranteed.
2. SVM supports Kernel, so we can model non-linear relations.

#### Weaknesses

1. Because of Kernel method, we need to write a good kernel function. Also, not any functions can be used as kernels. No one knows the best gamma or C is, so we have to try it as many times as possible to see values that may result a better outcome.
2. Data visualization. It is easier to draw and understand 2D (two x variables) SVM model, but we will never know how our model (57 dimensional) looks like.

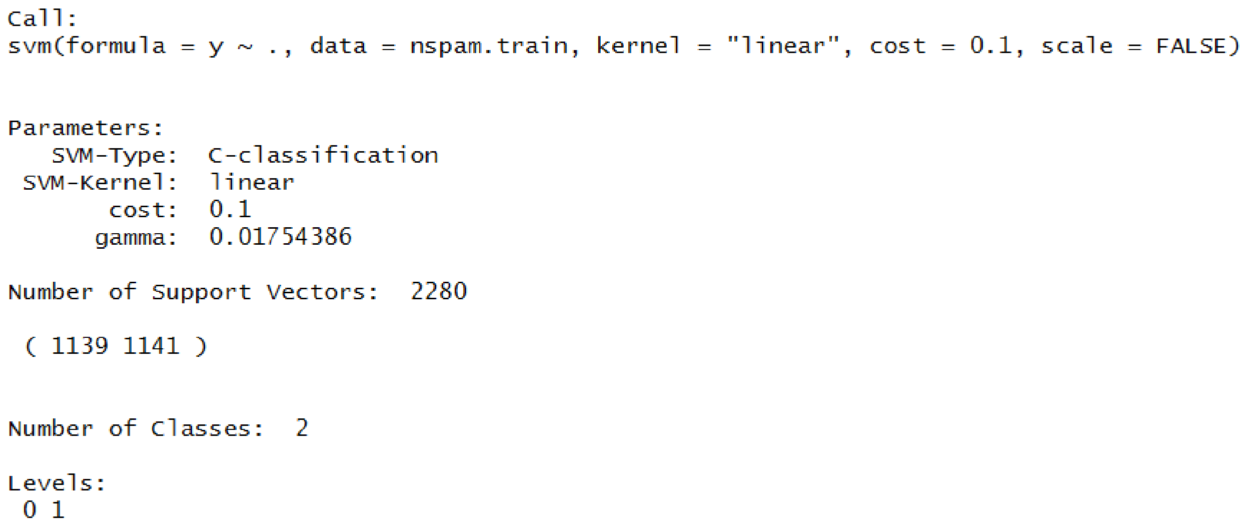
## Metrics

### Model selection

To build a SVM model, we first choose the linear kernel function to fit the data and we set the cost to 0.1.

Model 1 : Linear Kernel and cost = 0.1

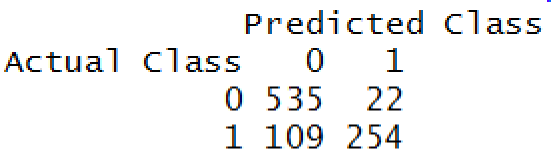
```
install.packages("e1071")
library(e1071)
svm.model <- svm(y ~ . , kernel="linear", cost= 0.1, scale=FALSE, data=spam.train)
summary(svm.model)
```



```
Call:
svm(formula = y ~ ., data = nspam.train, kernel = "linear", cost = 0.1, scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.1
      gamma:  0.01754386

Number of Support Vectors:  2280

 ( 1139 1141 )


Number of Classes:  2

Levels:
 0 1
```

After fitting the data with our first SVM model, we can use the predict function to predict the test data. Therefore, we can get the confusion matrix by the following code.

```
svm.prediction <- predict(svm.model, newdata = spam.test,type="class")
table(`Actual Class` = spam.test$y, `Predicted Class` = svm.prediction)
```



```
                Predicted Class
Actual Class    0   1
           0 535  22
           1 109 254
```

In this case, we need to pay attention to the false positive rate, which is mispredict the non-spam data (y=0) into spam data (y=1). Here we get 22 e-mails of 920 e-mails, and the false positive rate is **3.95%.** We are trying to lower this number. Moreover, we can also look at and try to increase the accuracy.

```
svm.error.rate <- sum(spam.test$y != svm.prediction)/nrow(spam.test)
print(paste0("Accuracy (Precision): ", 1 - svm.error.rate))
```

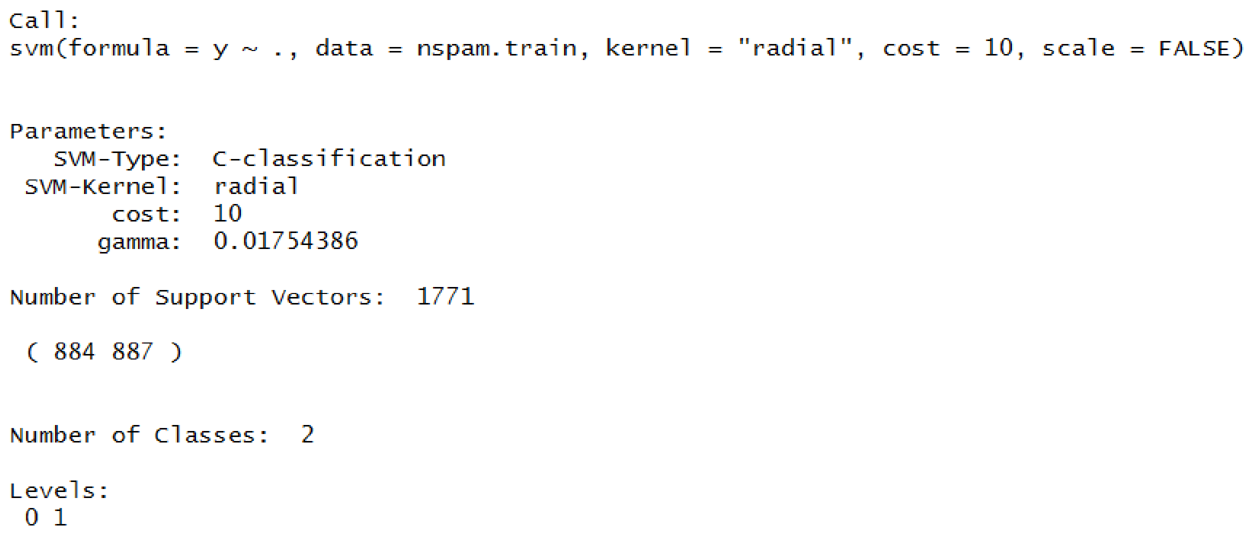
**"Accuracy (Precision): 0.857608695652174"**

Model 2 : Radial Kernel and cost = 10, number of support vectors = 1771

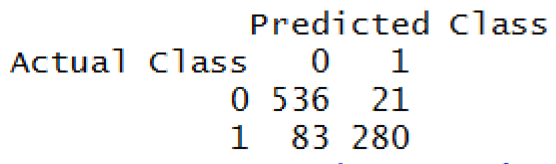
Next, try to fit SVM in a radial kernel

```
svm.model1 <- svm(y ~ . , kernel="radial", cost= 10, scale=FALSE, data=nspam.train)
```

summary(svm.model1)



```
Call:
svm(formula = y ~ ., data = nspam.train, kernel = "radial", cost = 10, scale = FALSE)


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

Number of Support Vectors:  1771

 ( 884 887 )


Number of Classes:  2

Levels:
 0 1
```



```
                Predicted Class
Actual Class     0    1
             0 536   21
             1  83  280
```

**"Accuracy (Precision): 0.88695652173913"**
We can see that the the accuracy increase, thus the radial model would be a  proper way to measure this dataset.

Cross Validation:
Model 3 : Radial Kernel and cost = 10 with cross validation, number of support vectors = 865
After trying the two model, we choose to do the cross validation using the function tune() in R to help us find the best model.  In the tune function, set the range of cost into a list of cost=c(0.01,0.1,1,10). This way, the tune function will use the radial kernel to find a proper cost number from the range I set. In this analysis, it shows that the best c would be 10.

tuned <- tune(svm,y ~ . ,data=nspam.train, kernel="radial", ranges=list(cost=c(0.01,0.1,1,10)))
summary(tuned)

```
Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
   10

- best performance: 0.06737289

- Detailed performance results:
   cost      error  dispersion
1  0.01 0.28823347 0.023376471
2  0.10 0.09128152 0.005498062
3  1.00 0.07090256 0.007494054
4 10.00 0.06737289 0.010150467
```

To get the best model after cross validation, we can store the model from the data frame of tuned.
bestmodel <- tuned$best.model
summary(bestmodel)

```
Call:
best.tune(method = svm, train.x = y ~ ., data = nspam.train, ranges = list(cost = c(0.01, 0.1,
    1, 10)), kernel = "radial")


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

Number of Support Vectors:  865

 ( 414 451 )


Number of Classes:  2

Levels:
 0 1
```

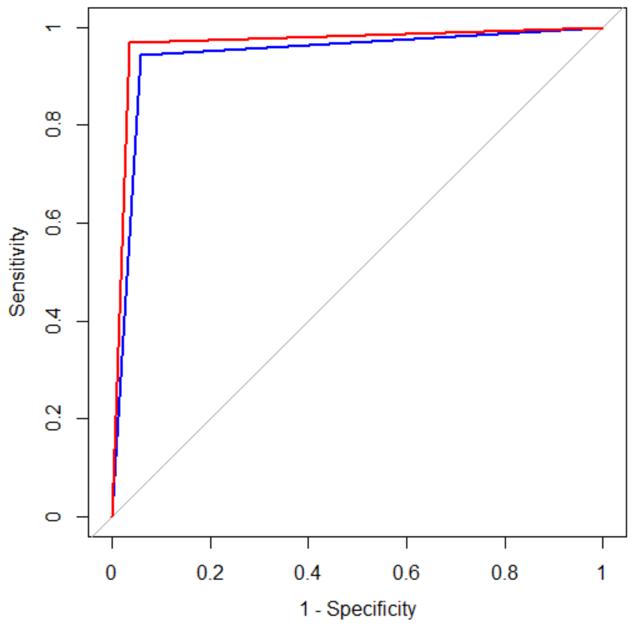
```
               Predicted Class
Actual Class    0   1
            0 537  20
            1  33 330
```

**"Accuracy (Precision): 0.942391304347826"**

With the model 3, we get the higher accuracy to 0.94 and now the false positive rate turn lower to **3.59%. Both results show that this model is better than the previous ones.**

ROC curve

The following graph shows the ROC curve of training and testing data. The red represent the training and the blue represent the testing. First, the AUC which is the area below the line is large. This means that this is a model with high accuracy. Secondly, it shows that the model fit better on training dataset, and this is reasonable. However, the two curve are close together almost on the same line, and thus there are no overfitting or underfitting problems. This means this SVM model is good on fitting the dataset.



## Testing the accuracy on training and testing data

In the section above, we tested the accuracy on the testing dataset. We would also like to see the accuracy on the training dataset. We use the model 3 which was concluded as the best model.

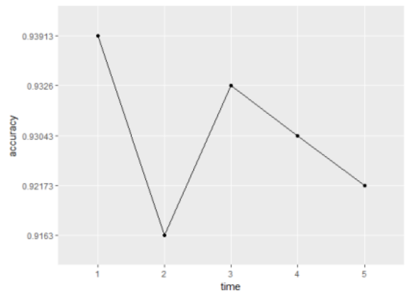
```
svm.prediction_1 <- predict(bestmodel, newdata = spam.train,type="class")
table(`Actual Class` = spam.train$y, `Predicted Class` = svm.prediction_1)
 svm.error.rate1 <- sum(spam.train$y != svm.prediction_1)/nrow(spam.train)
print(paste0("Accuracy (Precision): ", 1 - svm.error.rate1))
```

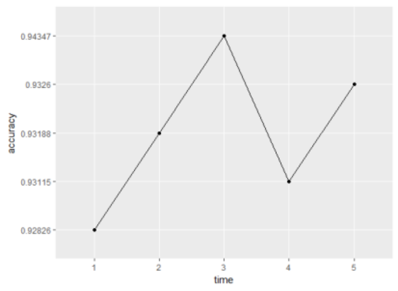
```
             Predicted Class
Actual Class    0    1
           0 2185   46
           1   83 1367
```

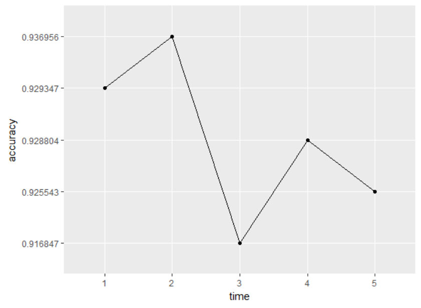
**"Accuracy (Precision): 0.964955175224124"**

The accuracy is 96% for training dataset. Compared the 94% accuracy for the testing data, it is normal that the accuracy of training data is higher. However, since the two numbers don't have a large difference, it may imply that this model doesn't have the overfitting problem.

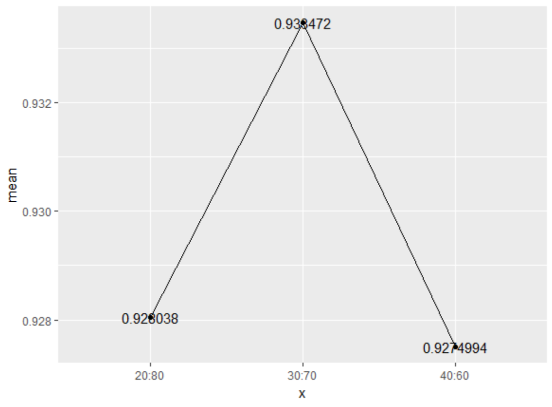
## Impact of training data size on test accuracy







**20:80** **30:70** **40:60**



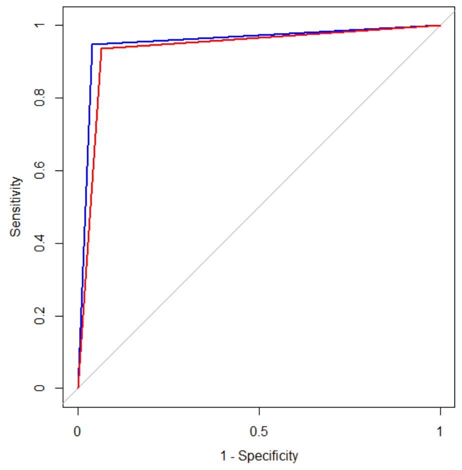
I randomly run the accuracy five times for each in three different size of training and testing data. First, 30:70 has the highest accuracy. Thus it may be in this classifier, choosing 30:70 would be a better choice. Second, in 30:70 way of splitting data, it shows more stable or tend to toward stationary than other two. With less training dataset to train the model, the accuracy fluctuate a lot. This can be seen in 40:60 graph.

## Redo 30:70 with the best SVM model

As we redo the model fitting in 30:70 size of splitting data, we get a higher accuracy to **95% t**han 94% before. There are 26 e-mails that are classified wrong and the false positive rate is **3.00%**. Since there is only a 1% difference, the ROC curve doesn't show too much difference comparing with the 20:80 size of fitting model. These two ROC curve present that the model are almost quite accurate to predict and measure the dataset.

```
             Predicted Class
Actual Class    0   1
            0 840  26
            1  36 478
```

**"Accuary (Precision): 0.955072463768116"**



## Conclusion

Both of the two classifiers can help us predict if an email is spam quite successfully with the information of word frequencies and character occurrence. The Naive Bayes model could achieve an accuracy of 85.80% on the test set with a train:test split of 75:25. The Support Vector Machine model could achieve an accuracy of 95.51% on the test set with a train:test split of 70:30. In the case of email filtering, we take particular attention to the False Positive Rate, because it could bring much more troubles to mistake a normal email as spam than the reverse. Thus, we calculate the False Positive Rate (FPR) every time we run each model and favor models with particularly low FPR. However, it's hard for a model to achieve the highest accuracy and the lowest FPR at the same time, so we need to make trade-offs between the two metrics. Overall, the SVM classifier could achieve better accuracy and thus is a better model. The different split of train and test set has an effect on the classifiers' performances, and each classifier has their respective optimal split ratio. When building a spam classifier, it's important to test various models, validate to get the best parameters, calculate different metrics, and try different train:test split ratios. This way, we believe we would have an optimal classifying machine learning model.

## Bonus - Decision Tree

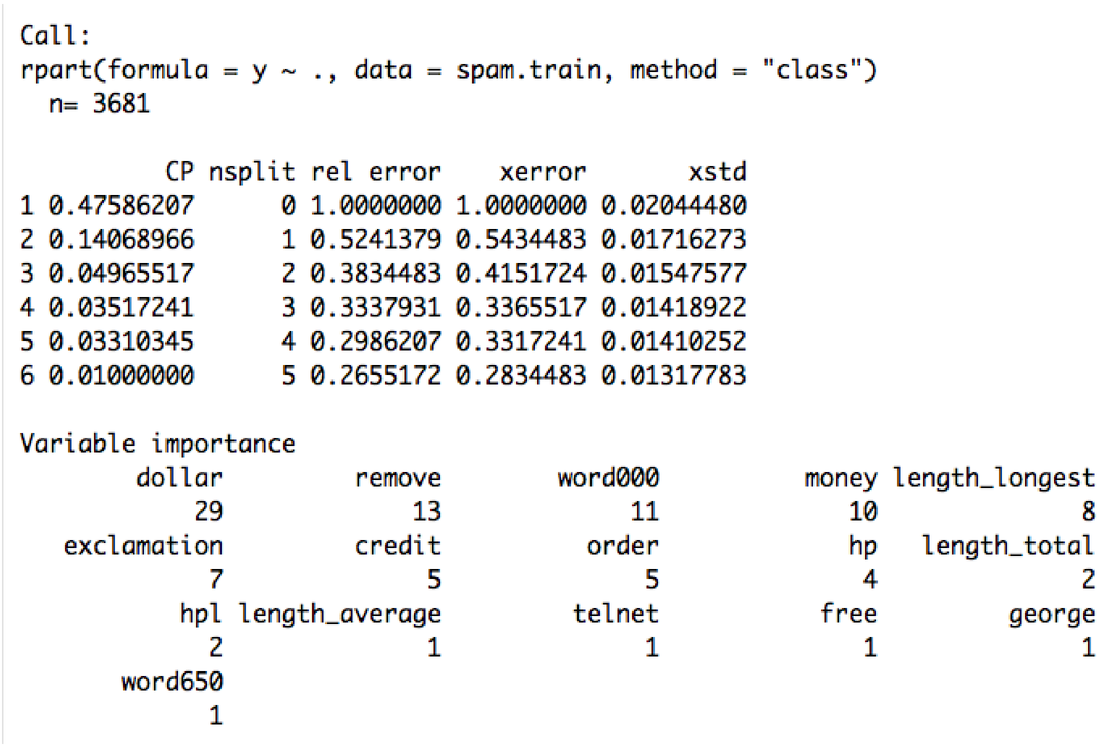
### Model Introduction

If we want to sketch the decision tree, we will start with the entire population (in this case, it is training data.). Given any one predictor (word_freq_make, etc), the population can be partitioned into discrete subset. Then, find the predictor that produces that "purest" subsets, i.e. most differentiated probabilities of response. Next, keep splitting subsets into subsets. When we stop splitting, the entire population is divided into a number of subsets, each of which has a probability of the response (e.g. probability of spam)

1. Build decision tree model

To build a decision tree, we need rpart package in R.

tree.model <- rpart(y ~ ., method="class", data=spam.train)

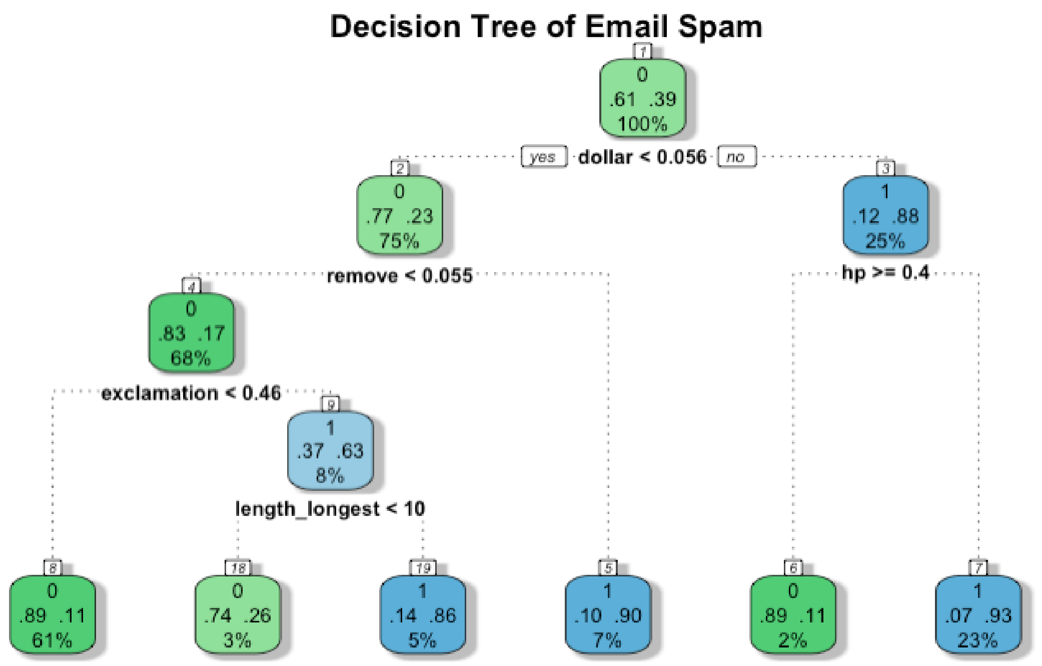


```
Call:
rpart(formula = y ~ ., data = spam.train, method = "class")
  n= 3681

          CP nsplit rel error    xerror       xstd
1 0.47586207      0 1.0000000 1.0000000 0.02044480
2 0.14068966      1 0.5241379 0.5434483 0.01716273
3 0.04965517      2 0.3834483 0.4151724 0.01547577
4 0.03517241      3 0.3337931 0.3365517 0.01418922
5 0.03310345      4 0.2986207 0.3317241 0.01410252
6 0.01000000      5 0.2655172 0.2834483 0.01317783

Variable importance
         dollar          remove         word000          money length_longest
             29              13              11             10              8
    exclamation          credit           order             hp   length_total
              7               5               5              4              2
            hpl  length_average          telnet           free         george
              2               1               1              1              1
        word650
              1
```

The summary shows 16 variables that affect spam or not spam.

This summary is super long, because R analyzes every predictor in every node, every split. Therefore, we use rattle, rpart.plot, and RColorBrewer packages and the following code to draw a decision tree.

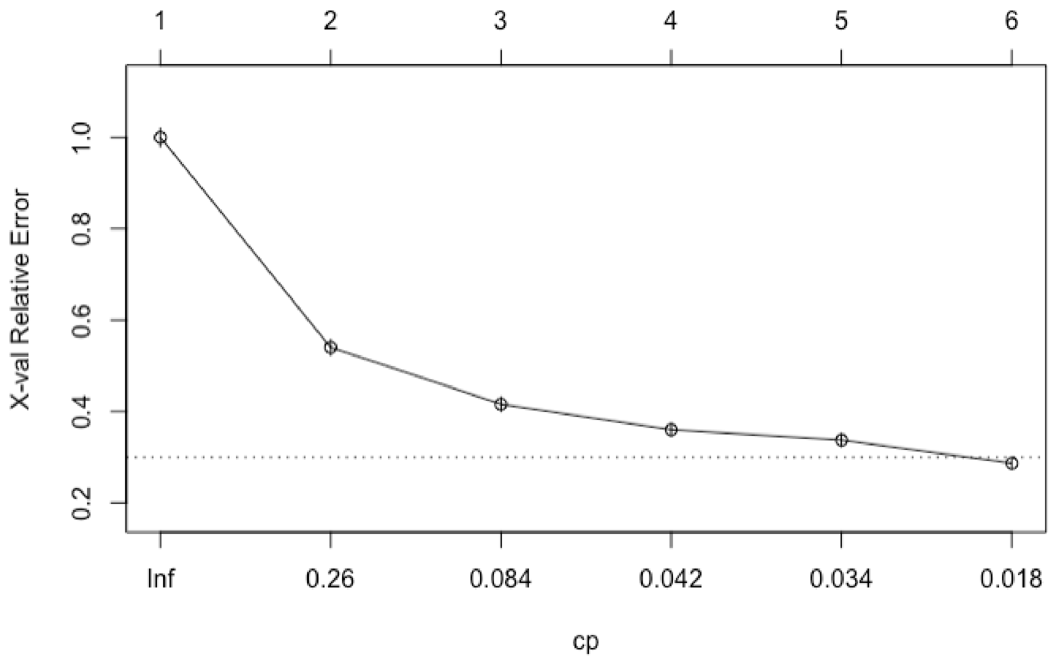
fancyRpartPlot(tree.model)



To find out how the tree performs, we use printcp(tree.model)

```
Classification tree:
rpart(formula = y ~ ., data = spam.train, method = "class")

Variables actually used in tree construction:
[1] dollar         exclamation    hp             length_longest remove

Root node error: 1450/3681 = 0.39391

n= 3681

        CP nsplit rel error  xerror     xstd
1 0.475862      0   1.00000 1.00000 0.020445
2 0.140690      1   0.52414 0.54000 0.017123
3 0.049655      2   0.38345 0.41586 0.015486
4 0.035172      3   0.33379 0.36000 0.014597
5 0.033103      4   0.29862 0.33724 0.014202
6 0.010000      5   0.26552 0.28621 0.013234
```

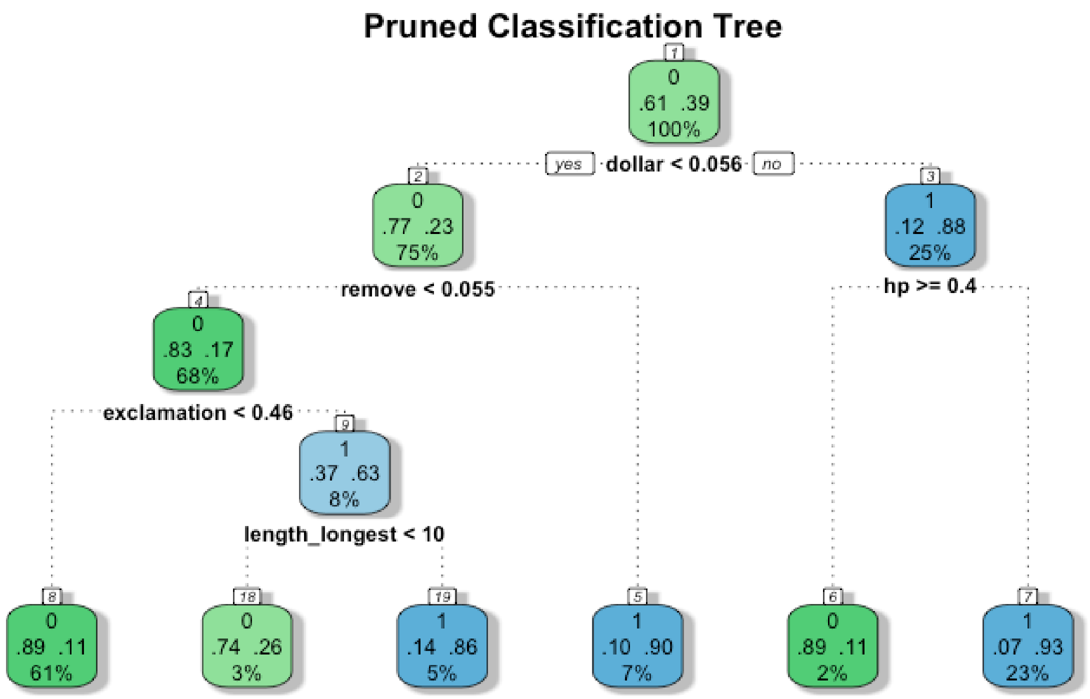
In order to get a better model, we need to prune a predictor that has the least cross validation error. Then, we use plotcp(tree.model) to check cross validation error summary in a graph. *The cp values are plotted against the geometric mean to depict the deviation until the minimum value is reached.*



Then, we prune the tree with

```
ptree<- prune(tree.model,
        cp=tree.model$cptable[which.min(tree.model$cptable[,"xerror"]),"CP"])
fancyRpartPlot(ptree, uniform=TRUE,
         main="Pruned Classification Tree")
```

In this code, `which.min(tree.model$cptable[,"xerror"]),"CP"` reduces the optimal cp value associated with the minimum error.



## Metrics

Firstly, we are going to check the accuracy on the test testing data with the following code.

```
tree.prediction <- predict(ptree, newdata = spam.test, type = "class")
table(`Actual Class` = spam.test$y, `Predicted Class` = tree.prediction)
tree.error.rate <- sum(spam.test$y != tree.prediction)/nrow(spam.test)
print(paste0("Accuracy (Precision): ", 1 - tree.error.rate))
```

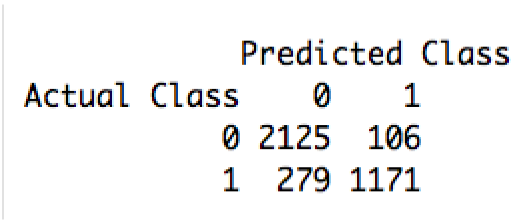
```
            Predicted Class
Actual Class   0   1
           0 530  27
           1  64 299
```

**"Accuracy (Precision): 0.901086956521739"**

We get an accuracy to 0.90 for testing data.

Then, we calculate the accuracy on the training data, and compare it with the one with testing accuracy in order to check if there is overfitting or underfitting problem.

```
tree.prediction1 <- predict(ptree, newdata = spam.train, type = "class")
table(`Actual Class` = spam.train$y, `Predicted Class` = tree.prediction1)
tree.error.rate1 <- sum(spam.train$y != tree.prediction1)/nrow(spam.train)
print(paste0("Accuracy (Precision): ", 1 - tree.error.rate1))
```



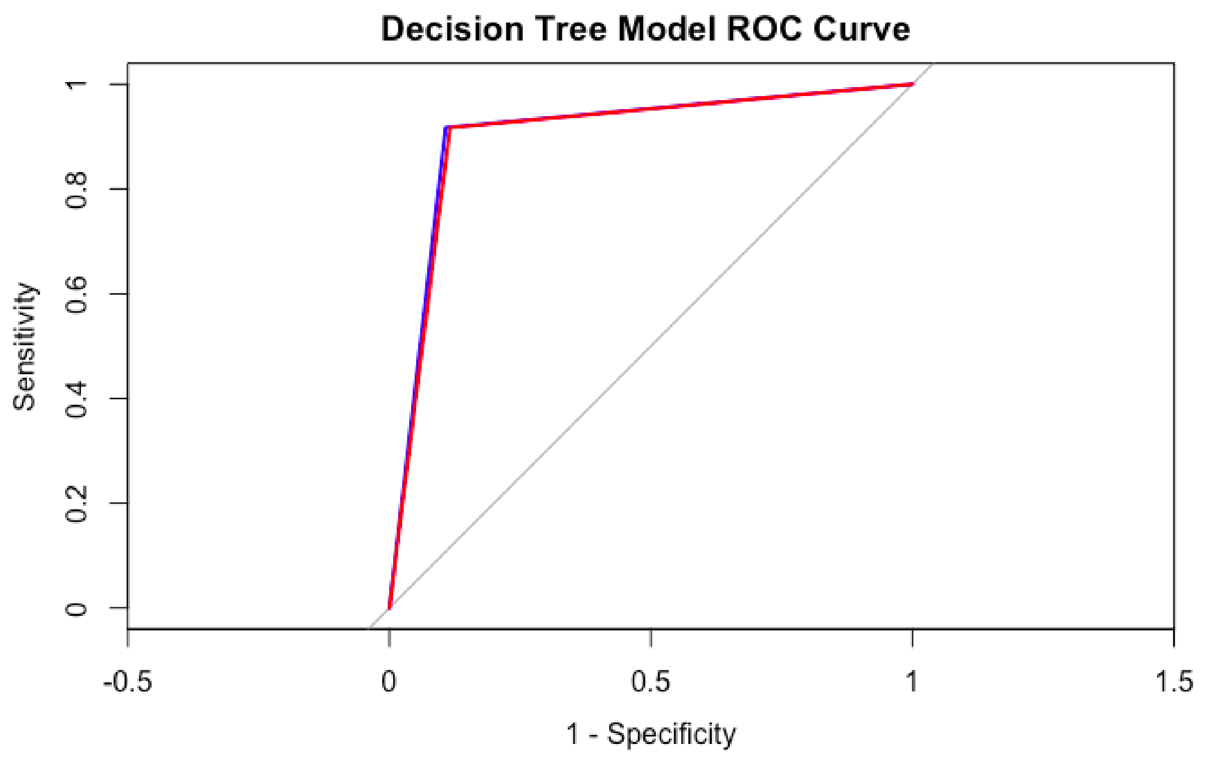
```
               Predicted Class
Actual Class     0    1
             0 2125  106
             1  279 1171
```

**"Accuracy (Precision): 0.895408856289052"**
Although this value is a bit lower than testing accuracy, they are nearly the same. Therefore, two numbers don’t have a large difference, it implies that this model doesn’t have the overfitting or underfitting problem.

Then, we check the ROC curve on the testing set and the training set respectively.

```
ROC<-roc(as.numeric(tree.prediction),as.numeric(spam.test$y))
ROC
par("mar")
par(mar=c(1,1,1,1))

ROC1<-roc(as.numeric(tree.prediction1),as.numeric(spam.train$y))
ROC1
par(mar=c(1,1,1,1))

{plot(ROC,col="blue",legacy.axes = TRUE, main="Decision Tree Model ROC Curve ")
 lines(ROC1,col="red",legacy.axes = TRUE)}
```

ROC is for the testing data (blue line), and ROC1 is for the training data (red line). AUC of ROC is 0.9047, and AUC of ROC1 is 0.9005. These two areas are similar, so we get the following graph.



The graph shows that "*the closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the test.*".

**Resource**

A. (2012, July 21). Why split data in the ratio 70:30? Retrieved March 17, 2017, from http://information-gain.blogspot.com/2012/07/why-split-data-in-ratio-7030.html

Deshpande, B. (2012, January 19). Beware of 2 facts when using Naive Bayes classification for analytics. Retrieved March 17, 2017, from http://www.simafore.com/blog/bid/100934/Beware-of-2-facts-when-using-Naive-Bayes-classification-for-analytics

Deshpande, B. (2012, July 24). 2 ways of using Naive Bayes classification for numeric attributes. Retrieved March 17, 2017, from http://www.simafore.com/blog/bid/107702/2-ways-of-using-Naive-Bayes-classification-for-numeric-attributes

Eric Kim(2013, September 1),Everything You Wanted to Know about the Kernel Trick, Retrieved March 17, 2017, from http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html

Lab 5: Naive Bayes: Toy example. (n.d.). Retrieved March 17, 2017, from http://rpubs.com/dvorakt/144238

Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., Leisch, F., Chang, C., & Lin, C. (2017, February 2). Package 'e1071'. Retrieved March 17, 2017, from https://cran.r-project.org/web/packages/e1071/e1071.pdf

O. (2013, December 11). Area Under Curve (AUC) - pROC package. Retrieved March 17, 2017, from http://myrcodes.blogspot.com/2013/12/area-under-curve-auc-proc-package.html

Peterson, E. (2013, June 6). Naive Bayes on continuous variables. Retrieved March 17, 2017, from http://stats.stackexchange.com/questions/61034/naive-bayes-on-continuous-variables

Renata Ghisloti Duarte Souza(2015, December),SVM in Practice, Retrieved March 17, 2017, from http://www.datasciencecentral.com/profiles/blogs/svm-in-practice

Sebastián Maldonado, Richard Weber, Jayanta Basak(2011), ,Information Sciences 181 (115–128) http://www.dii.uchile.cl/wp-content/uploads/2011/05/Simultaneos-feature-selection-and-classification-using-kernel-penalized-support-vector-machines1.pdf

Strickland, J. (2015, July 16). Naïve Bayes using R. Retrieved March 17, 2017, from https://bicorner.com/2015/07/16/naive-bayes-using-r/

Support Vector Machines (SVM). (n.d.). Retrieved March 17, 2017, from http://www.statsoft.com/Textbook/Support-Vector-Machines

Sharma, A. (2015, January 29). Creating, Validating and Pruning the Decision Tree in R. Retrieved March 17, 2017, from https://www.edureka.co/blog/implementation-of-decision-tree/

Littman (n.d.). Introduction to Support Vector Machines. Retrieved March 17, 2017, from https://www.cs.rutgers.edu/~mlittman/courses/ml04/svm.pdf

Auria Laura, Moro Rouslan A. (2008, August) Support Vector Machines (SVM) as a technique for solvency analysis, DIW Discussion Papers, No. 811. Retrieved March 17, 2017, from https://www.econstor.eu/bitstream/10419/27334/1/576821438.PDF

Y'barbo, D. (2010, August 13). Ways to improve the accuracy of a Naive Bayes Classifier? Retrieved March 17, 2017, from http://stackoverflow.com/questions/3473612/ways-to-improve-the-accuracy-of-a-naive-bayes-classifier