HW 3: Ensemble Techniques

Atharva Bhagwat

December 5, 2021

CS-GY 6923: MACHINE LEARNING
Professor: Dr. Raman Kannan

Contents

Chapter 1

Overview of the Dataset

The dataset is called "Airlines Codrna Adult" and was taken from OpenML.org.

The dataset consists of 29 independent variables and 1 dependent variable and has 1,076,790 observations. The dependent variable is called 'Delay' and is a categorical variable with only two values, 0 or 1. Therefore, this is a binary classification problem. The dataset is a CSV file and will be worked on using R.

Chapter 2

Loading the Dataset

The dataset CSV file is named 'AirlinesCodrnaAdult.csv' and is located inside 'dataset' folder. It is loaded into a variable named *data*.

```
# Function to Load Dataset
load_dataset = function(csvfile){read.csv(csvfile)}

cat("Loading Dataset...")
data = load_dataset('/home/2021/nyu/fall/acb9244/dataset/AirlinesCodrnaAdult.csv')      # Load Data
```

We will perform the following checks to see if the data is loaded properly.

1. Display the first few entries using the command head(data).
2. Check number of observations and number of features using dim(data.
3. Display the column names using the command names(data).

```
> head(data)
  Airline Flight AirportFrom AirportTo DayOfWeek Time Length codrna_X1
1      US   1558         PHX       CLT         3   15    222  0.698608
2      AA   2400         LAX       DFW         3   20    165  0.698608
3      AA   2466         SFO       DFW         3   20    195  0.698608
4      AS    108         ANC       SEA         3   30    202  0.698608
5      CO   1094         LAX       IAH         3   30    181  0.698608
6      DL   1768         LAX       MSP         3   30    220  0.698608
  codrna_X2 codrna_X3 codrna_X4 codrna_X5 codrna_X6 codrna_X7 codrna_X8 age
1  0.136363  0.281746  0.286873  0.599079  0.204906  0.569492  0.587408   2
2  0.136363  0.281746  0.286873  0.599079  0.204906  0.569492  0.587408   2
3  0.136363  0.281746  0.286873  0.599079  0.204906  0.569492  0.587408   2
4  0.136363  0.281746  0.286873  0.599079  0.204906  0.569492  0.587408   2
5  0.136363  0.281746  0.286873  0.599079  0.204906  0.569492  0.587408   2
6  0.136363  0.281746  0.286873  0.599079  0.204906  0.569492  0.587408   2
  workclass fnlwgt education education.num marital.status   occupation
1 State-gov  77516 Bachelors            13  Never-married Adm-clerical
2 State-gov  77516 Bachelors            13  Never-married Adm-clerical
3 State-gov  77516 Bachelors            13  Never-married Adm-clerical
4 State-gov  77516 Bachelors            13  Never-married Adm-clerical
5 State-gov  77516 Bachelors            13  Never-married Adm-clerical
6 State-gov  77516 Bachelors            13  Never-married Adm-clerical
   relationship  race  sex capitalgain capitalloss hoursperweek native.country
1 Not-in-family White Male           1           0            2  United-States
2 Not-in-family White Male           1           0            2  United-States
3 Not-in-family White Male           1           0            2  United-States
4 Not-in-family White Male           1           0            2  United-States
5 Not-in-family White Male           1           0            2  United-States
6 Not-in-family White Male           1           0            2  United-States
  Delay
1     1
2     1
3     1
4     0
5     1
6     0
```

```
> nrow(data)
[1] 1076790
> ncol(data)
[1] 30
```

```
> names(data)
 [1] "Airline"        "Flight"         "AirportFrom"    "AirportTo"
 [5] "DayOfWeek"      "Time"           "Length"         "codrna_X1"
 [9] "codrna_X2"      "codrna_X3"      "codrna_X4"      "codrna_X5"
[13] "codrna_X6"      "codrna_X7"      "codrna_X8"      "age"
[17] "workclass"      "fnlwgt"         "education"      "education.num"
[21] "marital.status" "occupation"     "relationship"   "race"
[25] "sex"            "capitalgain"    "capitalloss"    "hoursperweek"
[29] "native.country" "Delay"
```

We see that the dataset contains 1,076,790 observations and 30 features. We will further analyse the data in the next chapter.

Chapter 3

Exploratory Data Analysis

3.1 Understanding the Data

The first step to building machine learning models is understanding the data.

3.1.1 Target Variable

We will start with the target variable. We will look at number or classes, number of observations for each class and their percentage.

```
target_information <- function(dataframe){
       dataframe.target_unique_val <- unique(data$Delay)
       pprint('Class values in "Delay" column, ie: target variable',dataframe.target_unique_val)

       dataframe.target_unique_count <- table(data$Delay)
       pprint('Number of observations in each class',dataframe.target_unique_count)

       dataframe.target_unique_per <- (dataframe.target_unique_count/nrow(data))*100
       pprint('Percent of observations in each class',dataframe.target_unique_per)
}
```

```
> target_information()

Class values in "Delay" column, ie: target variable
[1] 1 0
Levels: 0 1

Number of observations in each class

     0      1
473652 603138

Percent of observations in each class

       0        1
43.98741 56.01259
```

The function target_information(), prints the unique class values, number of

observations for each class and their percentage. As we see only two values 0 and 1, we can say that this is a binary classification problem. We also observe that the classes are slightly imbalanced.

3.1.2 Types of Features

We have total of 29 features. A feature can be categorical or continuous. We can distinguish between such features just by looking at the values they hold.
Categorical Features: Airline, Flight, AirportFrom, AirportTo, DayOfWeek, age, workclass, education, education.num, marital.status, occupation, relationship, race, sex, hoursperweek, native.country.
Continuous Features: Time, Length, codrna_X1, codrna_X2, codrna_X3, codrna_X4, codrna_X5, codrna_X6, codrna_X7, codrna_X8, fnlwgt, capitalgain, capitalloss.
We can look at the datatype of all the features using sapply(data,class).

```
Columns and their class:
       Airline         Flight    AirportFrom      AirportTo      DayOfWeek
      "factor"      "integer"       "factor"       "factor"      "integer"
          Time         Length      codrna_X1      codrna_X2      codrna_X3
     "integer"      "integer"      "numeric"      "numeric"      "numeric"
     codrna_X4      codrna_X5      codrna_X6      codrna_X7      codrna_X8
     "numeric"      "numeric"      "numeric"      "numeric"      "numeric"
           age      workclass         fnlwgt      education  education.num
     "integer"       "factor"      "integer"       "factor"      "integer"
marital.status     occupation   relationship           race            sex
      "factor"       "factor"       "factor"       "factor"       "factor"
   capitalgain    capitalloss   hoursperweek native.country          Delay
     "integer"      "integer"      "integer"       "factor"       "factor"
```

3.2 Feature Analysis

We learnt that our features are of two types: categorical and continuous. A categorical feature is variable that contain a finite number of unique values. Where as a continuous features have infinite values between two values.

3.2.1 Categorical Features

Tests to run:

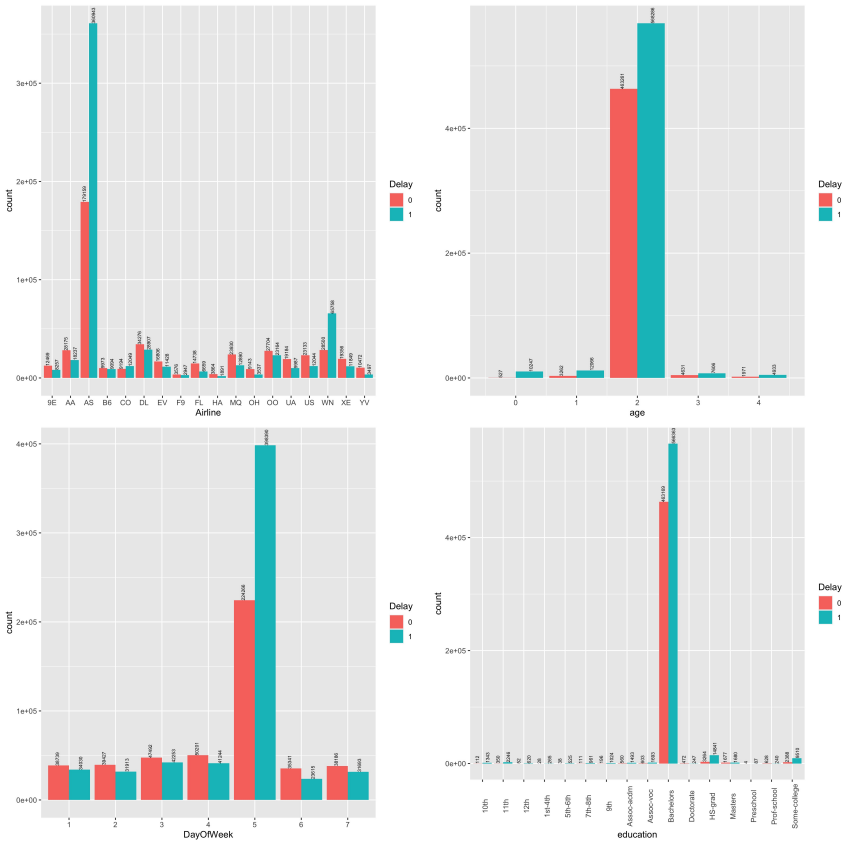
1. Check number of unique values.
2. Plot bar graphs to look at the distribution/percentage of unique values on target variable

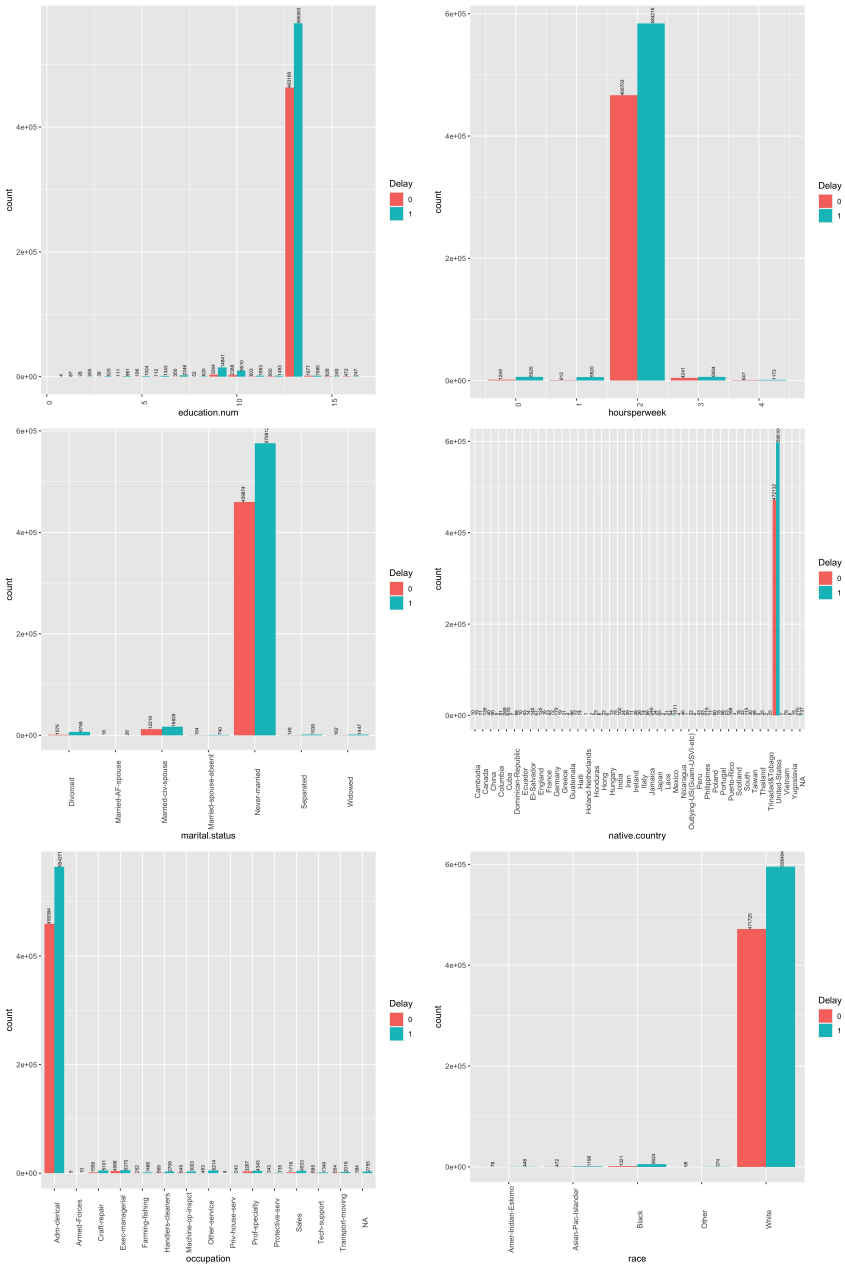
```
> unique_values <- function(){
+   for (column in categorical_features){
+     pprint(paste('Unique values in',column), table(data[column]))
+   }
+ }
```

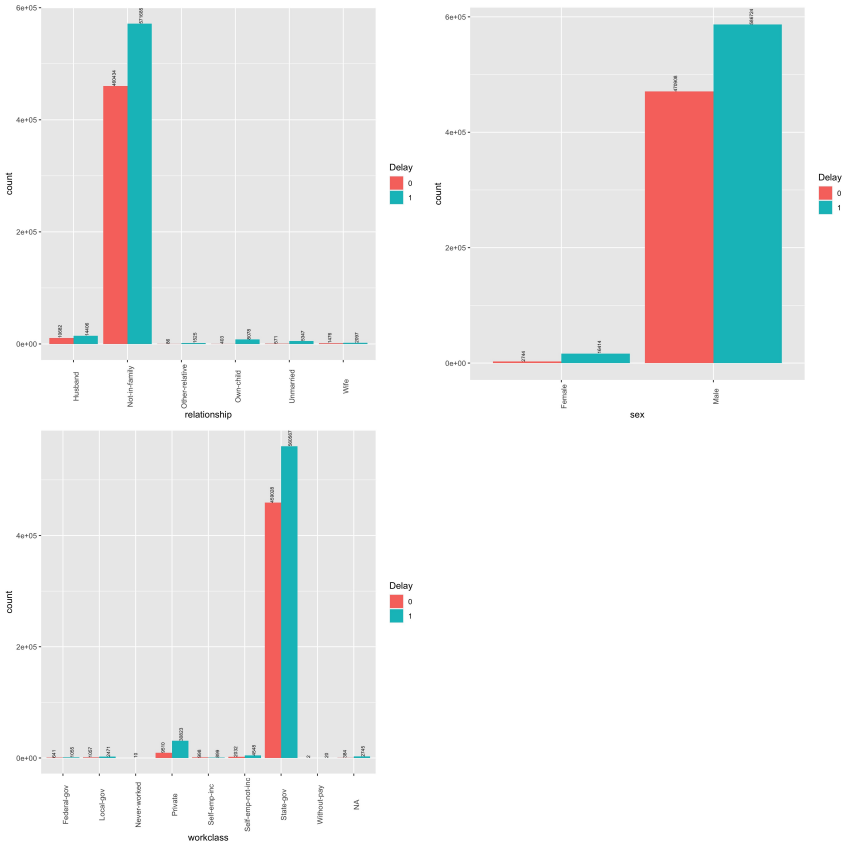
```
> probability_dist <- function(){
+   for (column in categorical_features){
+     pprint(paste('Percentage of unique values in',column),round(table(data[c(column,'Delay')])*100/nrow(data),digits=2))
+   }
+ }
```

We see that for every feature a single category has very large number of observations.

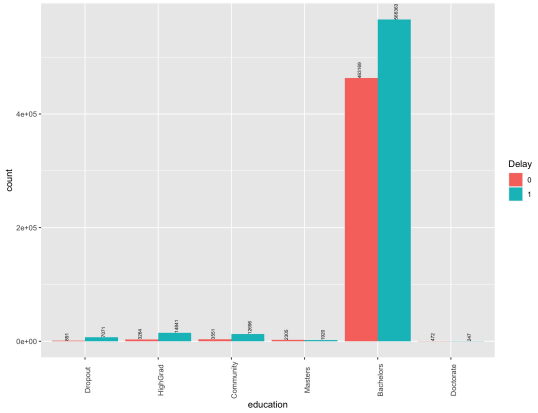
Distribution for character features:




count
education.num
hoursperweek
marital.status
native.country
occupation
race
Delay
0
1



From the above graphs we see that 'education' and 'education.num' deliver the same information. So we will drop column 'education.num' using data[, !names(data) %in% c('education.num'). We will also see that education has 16 unique values. We will recast this feature into broader groups using recode_factor(). We do this to group similar categories together to extract more information.



Old Label	New Label
Preschool	Dropout
1^{st}-4^{th}	Dropout
5^{th}-6^{th}	Dropout
7^{th}-8^{th}	Dropout
9^{th}	Dropout
10^{th}	Dropout
11^{th}	Dropout
12^{th}	Dropout
HS-grad	HighGrad
Some-college	Community
Assoc-acdm	Community
Assoc-voc	Community
Bachelors	Bachelors
Masters	Masters
Prof-school	Masters
Doctorate	Doctorate

3.2.2 Continuous Features

Tests to run:

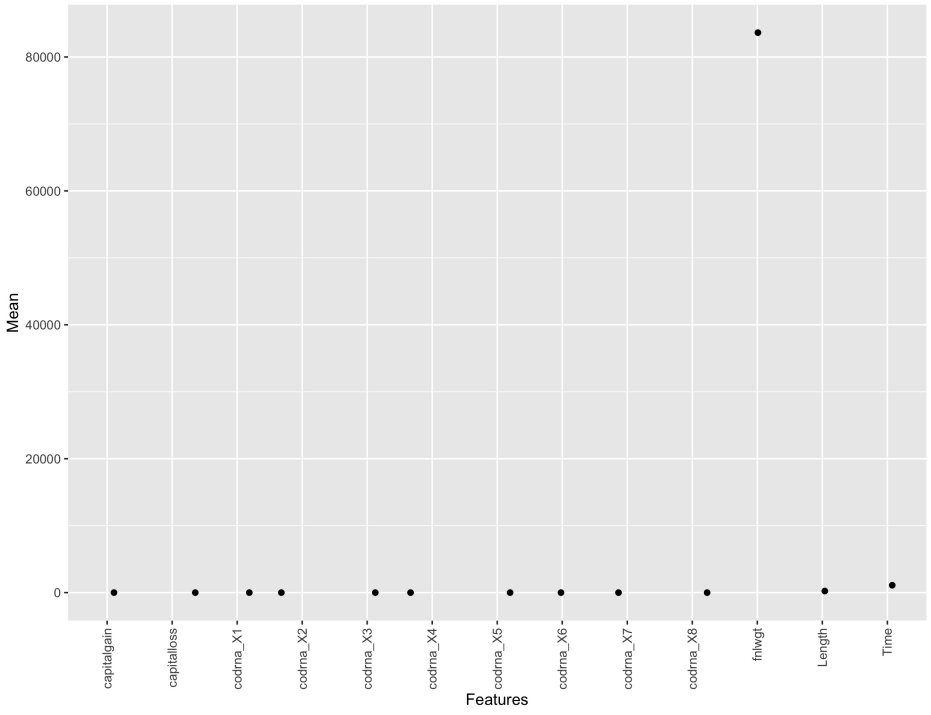
1. Calculate mean, median, mode, minimum, maximum, standard deviation, 1^{st} and 3^{rd} quartiles.

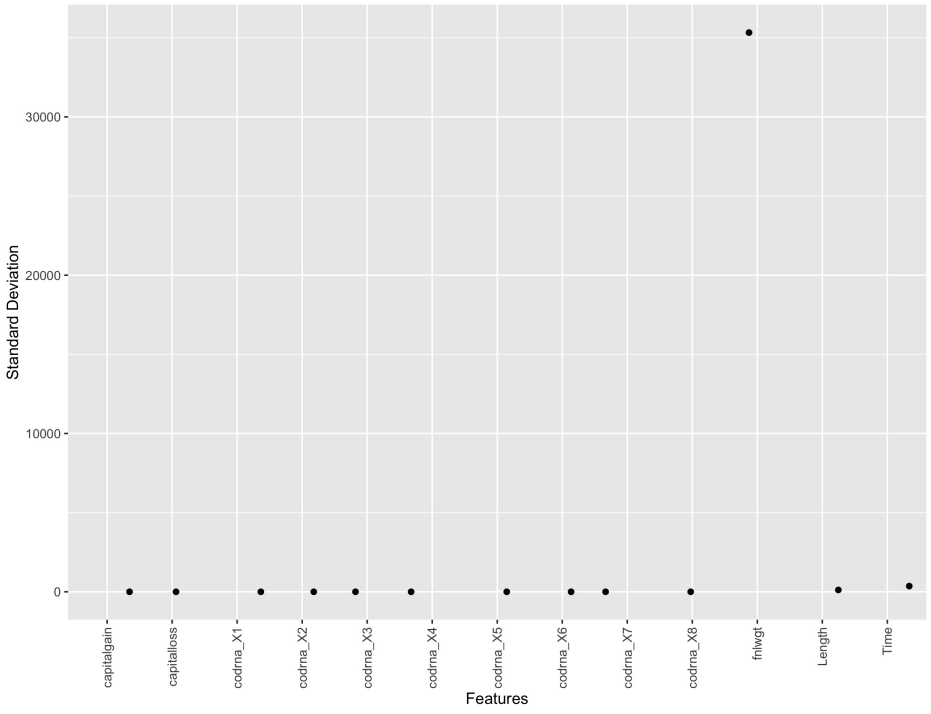
2. Check for outliers outliers.

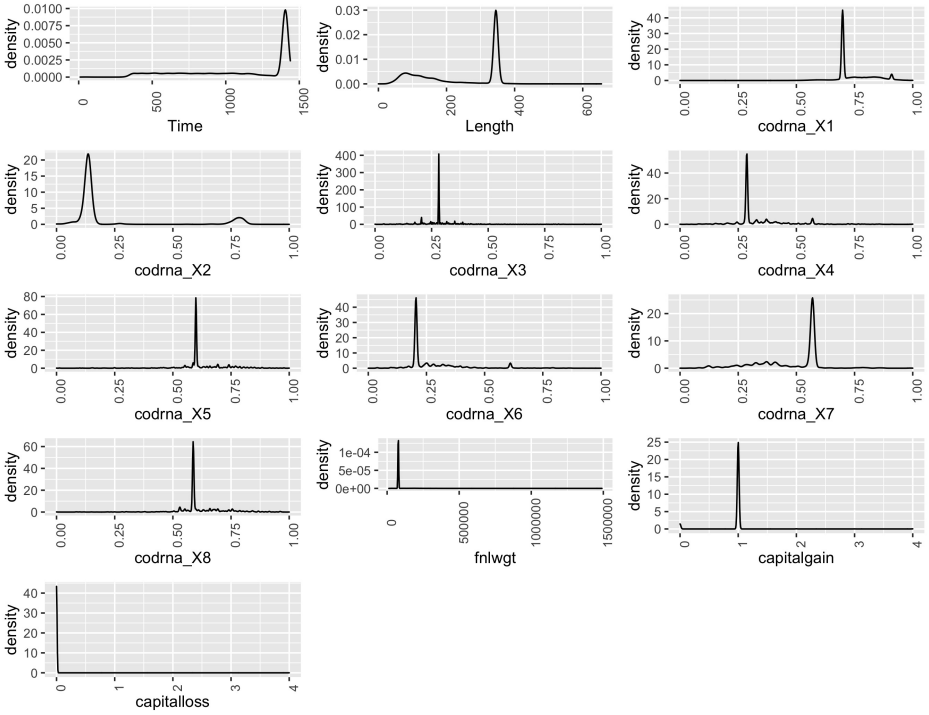
Many of these properties can be calculated using summary(data[continuous_features]). We calculate the outliers using the range $[Q1 - IQR * 1.5, Q3 + IQR * 1.5]$.

```
Summary:
      Time          Length        codrna_X1        codrna_X2
 Min.   :  10   Min.   :  0.0   Min.   :0.0000   Min.   :0.0000
 1st Qu.: 795   1st Qu.:115.0   1st Qu.:0.6986   1st Qu.:0.1364
 Median :1355   Median :345.0   Median :0.6986   Median :0.1364
 Mean   :1103   Mean   :237.2   Mean   :0.7453   Mean   :0.2219
 3rd Qu.:1405   3rd Qu.:345.0   3rd Qu.:0.8009   3rd Qu.:0.1364
 Max.   :1439   Max.   :655.0   Max.   :1.0000   Max.   :1.0000
   codrna_X3        codrna_X4        codrna_X5        codrna_X6
 Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
 1st Qu.:0.2744   1st Qu.:0.2869   1st Qu.:0.5991   1st Qu.:0.2049
 Median :0.2817   Median :0.2869   Median :0.5991   Median :0.2049
 Mean   :0.2907   Mean   :0.3410   Mean   :0.6204   Mean   :0.2709
 3rd Qu.:0.2817   3rd Qu.:0.3711   3rd Qu.:0.6590   3rd Qu.:0.3080
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
   codrna_X7        codrna_X8          fnlwgt         capitalgain
 Min.   :0.0000   Min.   :0.0000   Min.   :  12285   Min.   :0.0000
 1st Qu.:0.3711   1st Qu.:0.5874   1st Qu.:  77516   1st Qu.:1.0000
 Median :0.5695   Median :0.5874   Median :  77516   Median :1.0000
 Mean   :0.4763   Mean   :0.6145   Mean   :  83638   Mean   :0.9551
 3rd Qu.:0.5695   3rd Qu.:0.6590   3rd Qu.:  77516   3rd Qu.:1.0000
 Max.   :1.0000   Max.   :1.0000   Max.   :1490400   Max.   :4.0000
  capitalloss
 Min.   :0.000000
 1st Qu.:0.000000
 Median :0.000000
 Mean   :0.005733
 3rd Qu.:0.000000
 Max.   :4.000000
```

```
Standard deviation:
      Time      Length   codrna_X1   codrna_X2   codrna_X3   codrna_X4
    359.38      117.41        0.08        0.22        0.11        0.12
  codrna_X5   codrna_X6   codrna_X7   codrna_X8      fnlwgt capitalgain
      0.11        0.13        0.14        0.11    35324.01        0.25
capitalloss
      0.13
```


Mean
80000
60000
40000
20000
0
capitalgain
capitalloss
codrna_X1
codrna_X2
codrna_X3
codrna_X4
codrna_X5
codrna_X6
codrna_X7
codrna_X8
fnlwgt
Length
Time
Features


Standard Deviation
30000
20000
10000
0
capitalgain
capitalloss
codrna_X1
codrna_X2
codrna_X3
codrna_X4
codrna_X5
codrna_X6
codrna_X7
codrna_X8
fnlwgt
Length
Time
Features




```
> data.outliers <- c()
> for(column in continuous_features){
+   data_mean <- mean(data[,column])
+   data_sd <- sd(data[,column])
+   lower_cutoff <- data_mean-3*data_sd
+   higher_cutoff <- data_mean+3*data_sd
+   data.outliers_id <- which(data[,column]<lower_cutoff | data[,column]>higher_cutoff)
+   data.outliers <- c(data.outliers,data.outliers_id)
+ }
> data.outliers <- unique(data.outliers)
> pprint('Number of Outliers:',length(data.outliers))

Number of Outliers:
[1] 133877
> pprint('Percentage of Outliers:',round(length(data.outliers)*100/nrow(data),2))

Percentage of Outliers:
[1] 12.43
```

All continuous features have single peak. 12.43% of the total observations are outliers.

3.2.3 Missing Values

We inspect the data to see if there are any missing values.

```
missing_value_info <- function(dataframe){
        dataframe.missing_count <- sapply(dataframe, function(x) sum(is.na(x)))
        pprint('Number of missing values:',dataframe.missing_count)

        dataframe.missing_per <- (dataframe.missing_count/nrow(dataframe))*100
        pprint('Percentage of missing values:',dataframe.missing_per)
}
```

```
[> missing_value_info(data)

Number of missing values:
       Airline         Flight    AirportFrom      AirportTo      DayOfWeek
             0              0              0              0              0
          Time         Length      codrna_X1      codrna_X2      codrna_X3
             0              0              0              0              0
     codrna_X4      codrna_X5      codrna_X6      codrna_X7      codrna_X8
             0              0              0              0              0
           age      workclass         fnlwgt      education  education.num
             0           3129              0              0              0
marital.status     occupation   relationship           race            sex
             0           3139              0              0              0
   capitalgain    capitalloss   hoursperweek native.country          Delay
             0              0              0           1007              0

Percentage of missing values:
       Airline         Flight    AirportFrom      AirportTo      DayOfWeek
     0.0000000      0.0000000      0.0000000      0.0000000      0.0000000
          Time         Length      codrna_X1      codrna_X2      codrna_X3
     0.0000000      0.0000000      0.0000000      0.0000000      0.0000000
     codrna_X4      codrna_X5      codrna_X6      codrna_X7      codrna_X8
     0.0000000      0.0000000      0.0000000      0.0000000      0.0000000
           age      workclass         fnlwgt      education  education.num
     0.0000000      0.2905859      0.0000000      0.0000000      0.0000000
marital.status     occupation   relationship           race            sex
     0.0000000      0.2915146      0.0000000      0.0000000      0.0000000
   capitalgain    capitalloss   hoursperweek native.country          Delay
     0.0000000      0.0000000      0.0000000      0.0935187      0.0000000
```

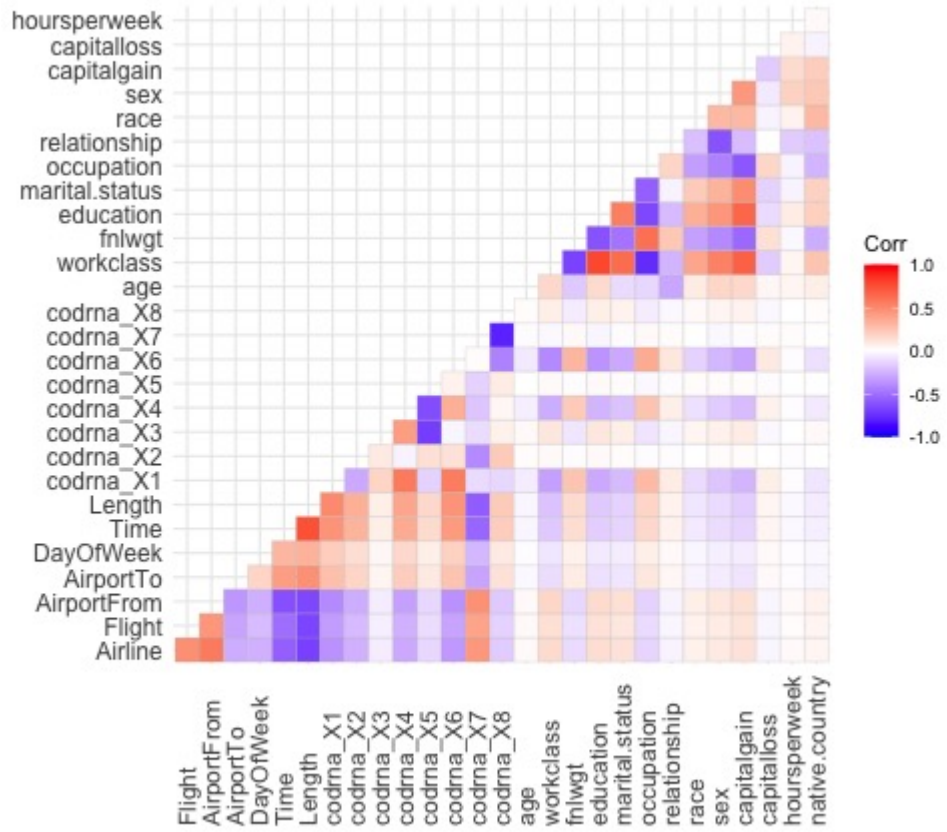
We see that 3 features: workclass (0.29%), occupation (0.29%), and native.country(0.09%) have missing values.

3.2.4 Handling Missing Values

Only three features: work class, occupation, and native country have missing values. The missing values are so small in number that we can just drop the observation with missing values using drop_na(data).

3.3 Correlation

As observed in figure not all features are of type numeric. We first convert these features to numeric so we can perform further steps. We use as.numeric(data[column]) to convert the features to numeric.
We now calculate the correlation using cor(data).



We plot the lower correlation matrix. Correlation values equal to 0 means that the variables are not correlated and values 1 and -1 mean that the variable are positively and negatively correlated respectively. After checking how many of the feature are highly correlated, ie: the absolute correlation coefficient > 0.5, we find 14 correlated features.

3.4 Eliminating Correlated Features

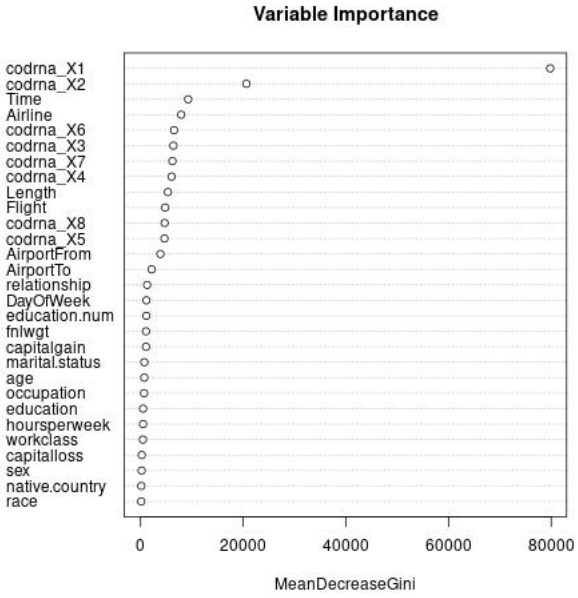
In 3.3 we saw that 14 features are highly correlated. We will eliminate these features as it would reduce the number of variables, reducing the model's complexity. We use a threshold of 0.5.

3.5 Variable Importance using Random Forest Classifier

The next step is to rank the features as per their importance. This is done using a Random Forest Classifier and the varImp() from the caret package.

To build a classifier, we divide the data into train and test sets. We will use 70% of data as the training set and 30% as testing test. We will randomize the dataset before splitting and will use set.seed(42) to replicate the result.

```
> set.seed(42)
> data.randomized <- data[sample(1:nrow(data),nrow(data)),]
>
> split_indexes <- sample(1:nrow(data),0.7*nrow(data),replace=F)
> data.train <- data.randomized[split_indexes,]
> data.test <- data.randomized[-split_indexes,]
>
> data.train_rf <- data.train
```



3.6 Variable Inflation Factor

VIF is a measure of multicollinearity among the variables. Multicollinearity is when two or more features are highly correlated to each other. This measure leads to higher variance in the model. To check VIF we will first convert the Delay variable to numeric as VIF can only be calculated for regression problems. We will use a generalized linear model to train.

```
Training GLM...> glm_model <- glm(Delay~., data.train_vif, family='binomial')
> glm_model_vif <- car::vif(glm_model)
> pprint('Variable Inflation Factor in ascending order:',sort(glm_model_vif))

Variable Inflation Factor in ascending order:
      age DayOfWeek education    Flight workclass   Airline codrna_X1
 1.021245  1.078048  1.171476  1.290806  1.302775  1.361159  1.414832
```

After removing the correlating features, we observe that the VIF values are not very high. This means that the chances of these set of features having high multicollinearity are low.

Chapter 4

EDA: Conclusion

4.1 Observations

- This is a binary classification problem.
- Dataset consists of both categorical and continuous features.
- Both categorical and continuous features have a single value with majority of observations.
- Feature 'education' is grouped into less number of categories to extract more information.
- Dataset contains very low number of missing values(0.6%). Hence the rows with missing values can be dropped.
- 14 features are highly correlated and should be dropped.
- Individual classifiers should be trained after correlated features are dropped and evaluated based on p-value, accuracy, recall, specificity and auc.

Chapter 5

Implementing Classifiers

The target variable for this dataset is 'Delay'. During application, it is crucial for the model to predict a +ve observation to be a +ve class. This is very similar to detection of cancer. So, for the purpose of analysis we will give preference to recall and precision values.

5.1 GLM

5.1.1 P-Value Analysis

P-values for all the features are less than 0.05. This implies that we can reject the null hypothesis. All the features have statistical relation with the target variable.

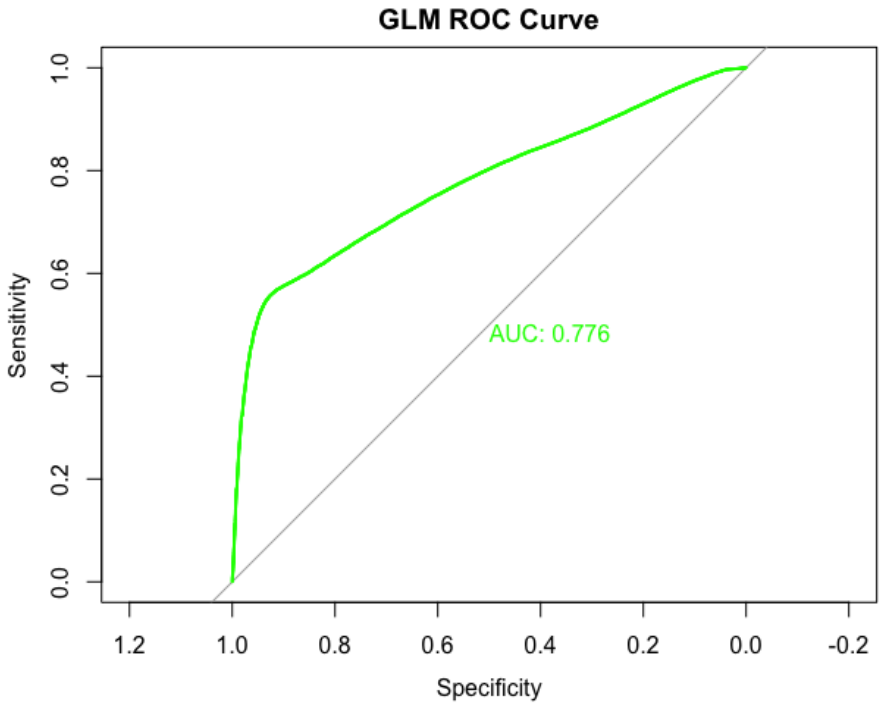
5.1.2 Confusion Matrix

The accuracy, recall(sensitivity), specificity and precision for both training and testing phase is very similar. However we do observe that the recall value for is lower than the precision. This means that the number of samples correctly classified as +ve might be low but the probability of the classification being correct is high.

```
GLM Summary:

Call:
glm(formula = Delay ~ ., family = "binomial", data = data.train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.9211  -0.9772   0.3186   0.8838   4.2328

Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept)    -1.763e+01  1.392e-01 -126.65  < 2e-16 ***
Airline         7.775e-02  6.009e-04  129.38  < 2e-16 ***
Flight          3.702e-05  1.568e-06   23.62  < 2e-16 ***
AirportTo      -1.105e-03  4.158e-05  -26.58  < 2e-16 ***
DayOfWeek      -9.450e-02  1.765e-03  -53.55  < 2e-16 ***
codrna_X1       2.244e+01  6.398e-02  350.69  < 2e-16 ***
codrna_X2       4.149e+00  1.820e-02  227.99  < 2e-16 ***
codrna_X3      -3.729e+00  3.051e-02 -122.25  < 2e-16 ***
age            -5.638e-01  1.356e-02  -41.59  < 2e-16 ***
workclass       4.952e-01  5.885e-03   84.14  < 2e-16 ***
relationship    3.500e-01  1.104e-02   31.70  < 2e-16 ***
race           -2.331e-01  2.036e-02  -11.45  < 2e-16 ***
capitalloss    -6.575e-01  2.388e-02  -27.54  < 2e-16 ***
hoursperweek   -7.319e-01  1.745e-02  -41.95  < 2e-16 ***
native.country  1.797e-02  2.197e-03    8.18 2.85e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1030307  on 750892  degrees of freedom
Residual deviance:  808650  on 750878  degrees of freedom
AIC: 808680

Number of Fisher Scoring iterations: 5
```

```
GLM Training Metrics:
Confusion Matrix and Statistics

          Reference
Prediction      0      1
         0 246529 138993
         1  84254 281117

               Accuracy : 0.7027
                 95% CI : (0.7017, 0.7037)
    No Information Rate : 0.5595
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4073

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.6692
            Specificity : 0.7453
         Pos Pred Value : 0.7694
         Neg Pred Value : 0.6395
             Prevalence : 0.5595
         Detection Rate : 0.3744
   Detection Prevalence : 0.4866
      Balanced Accuracy : 0.7072

       'Positive' Class : 1
```

```
GLM Testing Metrics:
Confusion Matrix and Statistics

          Reference
Prediction      0      1
         0 105842  59292
         1  36385 120293

               Accuracy : 0.7027
                 95% CI : (0.7011, 0.7043)
    No Information Rate : 0.558
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4072

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.6698
            Specificity : 0.7442
         Pos Pred Value : 0.7678
         Neg Pred Value : 0.6409
             Prevalence : 0.5580
         Detection Rate : 0.3738
   Detection Prevalence : 0.4869
      Balanced Accuracy : 0.7070

       'Positive' Class : 1
```



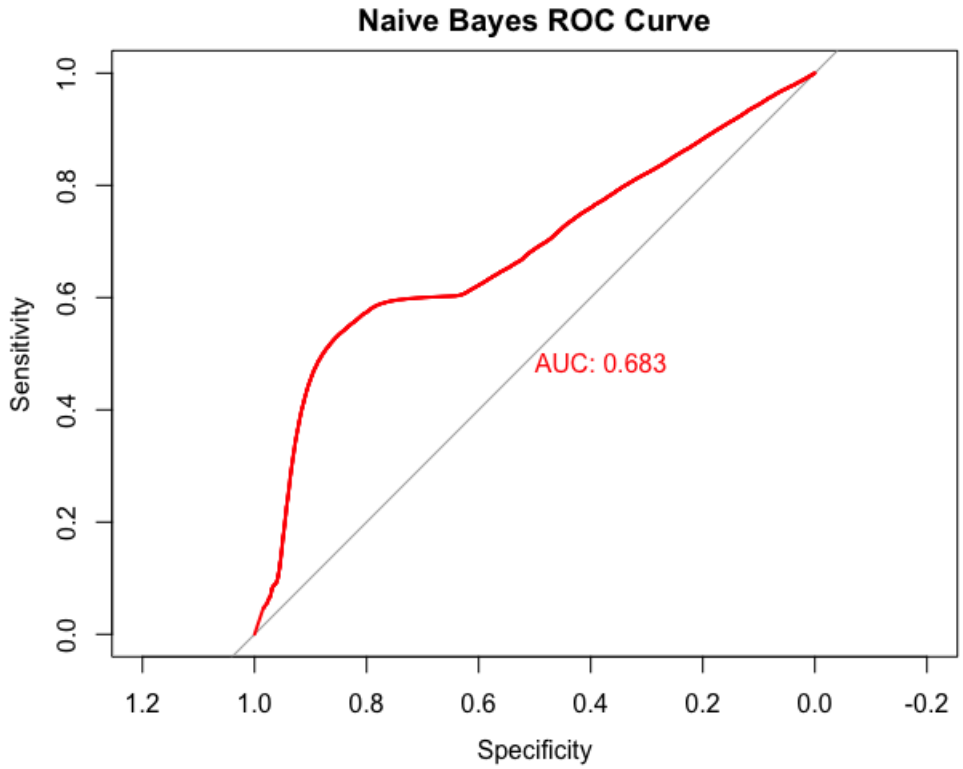
5.2 Naive Bayes

5.2.1 Confusion Matrix

For both the training and testing phase we see accuracy score, precision, recall and specificity scores to be very similar. But, recall value is too low and precision and specificity is high. Meaning the model is returning a positive class for only sample it is confident about and returning negative for the rest. Model has low accuracy of detecting +ve samples.

```
Naive Bayes Training Metrics:
Confusion Matrix and Statistics

          Reference
Prediction      0      1
         0 304848 263862
         1  25935 156248

               Accuracy : 0.6141
                 95% CI : (0.613, 0.6152)
    No Information Rate : 0.5595
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.2727

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.3719
            Specificity : 0.9216
         Pos Pred Value : 0.8576
         Neg Pred Value : 0.5360
             Prevalence : 0.5595
         Detection Rate : 0.2081
   Detection Prevalence : 0.2426
      Balanced Accuracy : 0.6468

       'Positive' Class : 1
```

```
Naive Bayes Testing Metrics:
Confusion Matrix and Statistics

          Reference
Prediction      0      1
         0 131047 112737
         1  11180  66848

               Accuracy : 0.6149
                 95% CI : (0.6133, 0.6166)
    No Information Rate : 0.558
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.2733

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.3722
            Specificity : 0.9214
         Pos Pred Value : 0.8567
         Neg Pred Value : 0.5376
             Prevalence : 0.5580
         Detection Rate : 0.2077
   Detection Prevalence : 0.2425
      Balanced Accuracy : 0.6468

       'Positive' Class : 1
```



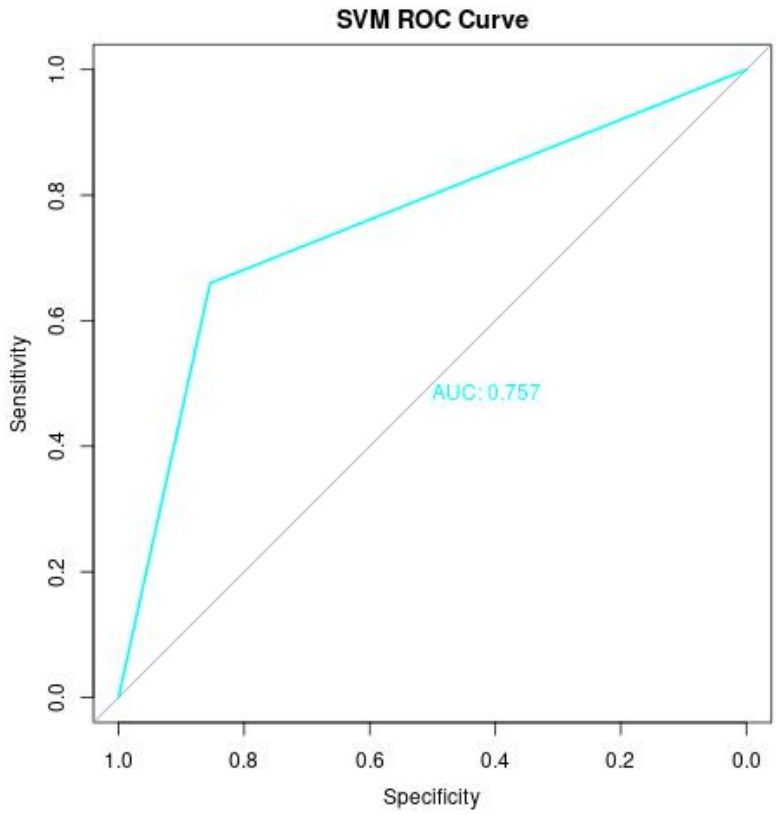
5.3 Support Vector Machines

5.3.1 Confusion Matrix

For both the training and testing phase we see accuracy score, precision, recall and specificity scores to be very similar. However, similar to GLM we observe that the recall value for is lower than the precision. SVM has a better precision value than GLM. It implies that the +ve observations correctly classified by SVM have a higher chance of actually being a +ve observation.

```
SVM Testing Metrics:
Confusion Matrix and Statistics

          Reference
Prediction      0      1
         0 121495  61101
         1  20732 118484

               Accuracy : 0.7457
                 95% CI : (0.7442, 0.7472)
    No Information Rate : 0.558
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4993

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.6598
            Specificity : 0.8542
         Pos Pred Value : 0.8511
         Neg Pred Value : 0.6654
             Prevalence : 0.5580
         Detection Rate : 0.3682
   Detection Prevalence : 0.4326
      Balanced Accuracy : 0.7570

       'Positive' Class : 1
```

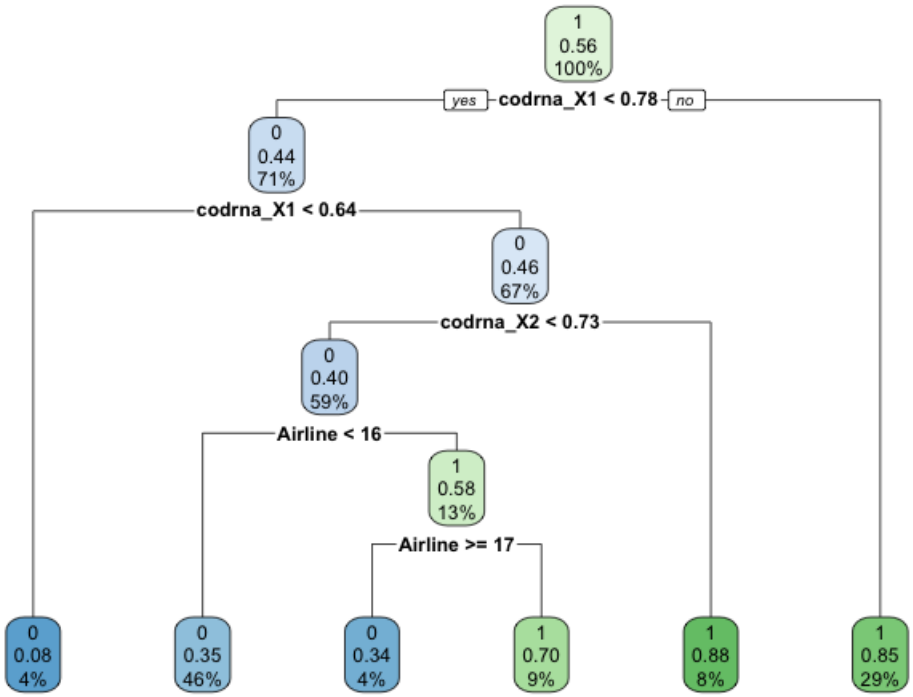
```
SVM Training Metrics:
Confusion Matrix and Statistics

          Reference
Prediction      0      1
         0 283007 142838
         1  47776 277272

               Accuracy : 0.7462
                 95% CI : (0.7452, 0.7471)
    No Information Rate : 0.5595
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5003

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.6600
            Specificity : 0.8556
         Pos Pred Value : 0.8530
         Neg Pred Value : 0.6646
             Prevalence : 0.5595
         Detection Rate : 0.3693
   Detection Prevalence : 0.4329
      Balanced Accuracy : 0.7578

       'Positive' Class : 1
```



5.4 Decision Tree

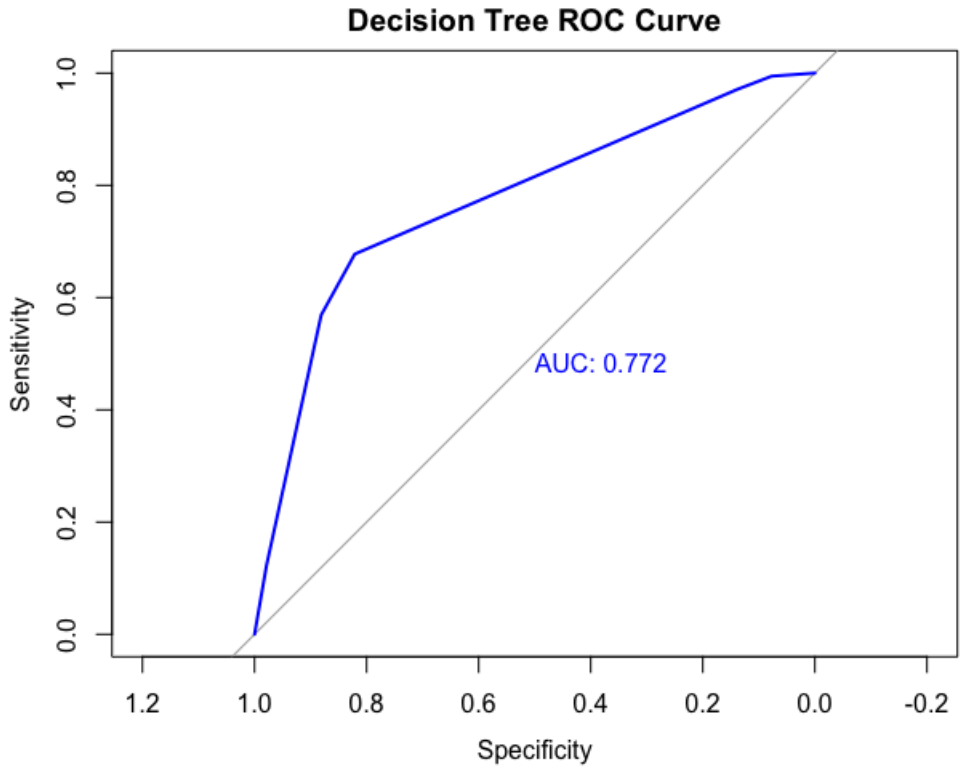
5.4.1 Confusion Matrix

For both the training and testing phase we see accuracy score, precision, recall and specificity scores to be very similar. But we see a slight increase in testing accuracy. This is a bad sign as the model is optimized for the training set. One of the reasons that this could have happened is because of the distribution of features in training and testing set. Applying a different seed might solve this problem.


1
0.56
100%
yes
codrna_X1 < 0.78
no
0
0.44
71%
codrna_X1 < 0.64
0
0.46
67%
codrna_X2 < 0.73
0
0.40
59%
Airline < 16
1
0.58
13%
Airline >= 17
0
0.08
4%
0
0.35
46%
0
0.34
4%
1
0.70
9%
1
0.88
8%
1
0.85
29%

```
Decision Tree Training Metrics:
Confusion Matrix and Statistics

          Reference
Prediction      0      1
         0 271424 136173
         1  59359 283937

               Accuracy : 0.7396
                 95% CI : (0.7386, 0.7406)
    No Information Rate : 0.5595
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4845

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.6759
            Specificity : 0.8206
         Pos Pred Value : 0.8271
         Neg Pred Value : 0.6659
             Prevalence : 0.5595
         Detection Rate : 0.3781
   Detection Prevalence : 0.4572
      Balanced Accuracy : 0.7482

       'Positive' Class : 1
```

```
Decision Tree Testing Metrics:
Confusion Matrix and Statistics

          Reference
Prediction      0      1
         0 116788  57916
         1  25439 121669

               Accuracy : 0.741
                 95% CI : (0.7395, 0.7425)
    No Information Rate : 0.558
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4871

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.6775
            Specificity : 0.8211
         Pos Pred Value : 0.8271
         Neg Pred Value : 0.6685
             Prevalence : 0.5580
         Detection Rate : 0.3781
   Detection Prevalence : 0.4571
      Balanced Accuracy : 0.7493

       'Positive' Class : 1
```



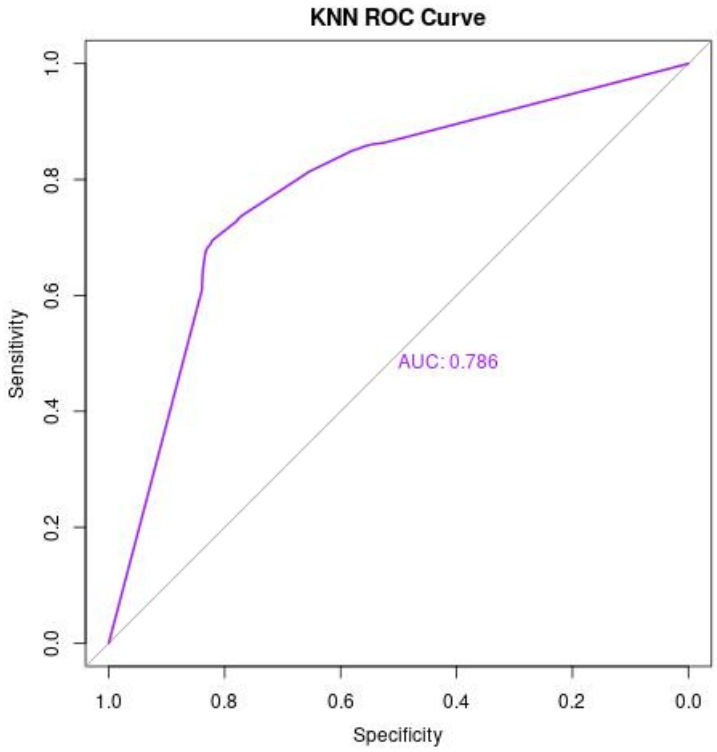
5.5 K Nearest Neighbor

5.5.1 Confusion Matrix

The testing accuracy has decreased by a 13%. As KNN is a low bias high variance model, has overfit on the training set. Altering the k value can solve this problem.

```
KNN Training Metrics:
Confusion Matrix and Statistics

          Reference
Prediction      0      1
         0 287876  47639
         1  42907 372471

               Accuracy : 0.8794
                 95% CI : (0.8787, 0.8802)
    No Information Rate : 0.5595
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7557

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.8866
            Specificity : 0.8703
         Pos Pred Value : 0.8967
         Neg Pred Value : 0.8580
             Prevalence : 0.5595
         Detection Rate : 0.4960
   Detection Prevalence : 0.5532
      Balanced Accuracy : 0.8784

       'Positive' Class : 1
```

```
KNN Testing Metrics:
Confusion Matrix and Statistics

          Reference
Prediction      0      1
         0 101777  40632
         1  40450 138953

               Accuracy : 0.748
                 95% CI : (0.7465, 0.7495)
    No Information Rate : 0.558
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.4893

 Mcnemar's Test P-Value : 0.525

            Sensitivity : 0.7737
            Specificity : 0.7156
         Pos Pred Value : 0.7745
         Neg Pred Value : 0.7147
             Prevalence : 0.5580
         Detection Rate : 0.4318
   Detection Prevalence : 0.5575
      Balanced Accuracy : 0.7447

       'Positive' Class : 1
```



5.6 Tabulating Model Performance on Test Set

This table is drawn with respect to model's performance on the testing set.

Model	Accuracy Score	Recall(Sensitivity)	Specificity	Precision	AUC
GLM	0.70	0.66	0.74	0.76	0.77
Naive Bayes	0.61	0.37	0.92	0.85	0.68
SVM	0.74	0.66	0.85	0.85	0.75
Decision Tree	0.74	0.67	0.82	0.82	0.77
KNN	0.74	0.77	0.71	0.77	0.78

5.7 Observations:

- Naive Bayes is the weakest performing model out of all the models. This might be because naive bayes is a high bias classifier. It assumes that all the features are independent of each other. This can lead to underfitting. Naive Bayes model has very low recall score (accuracy to detect +ve samples), as per the application of the dataset we can reject naive bayes model.

- SVM and Decision Tree are the top performing models. SVM and Decision Tree models are high-variance models, meaning large changes to training dataset/features which have high importance may cause the model to under perform as it might not understand underlying mapping between the features as good as say logistic regression.

Chapter 6

Ensemble Techniques

In this chapter we focus on computational part of modeling. Training models on large datasets consumes a lot of time. Techniques like cross validation and stacking can be implemented in parallel in order to reduce modeling time. R libraries used for parallel implementation: foreach and doParallel. We register all available cores using: registerDoParallel(detectCores()).

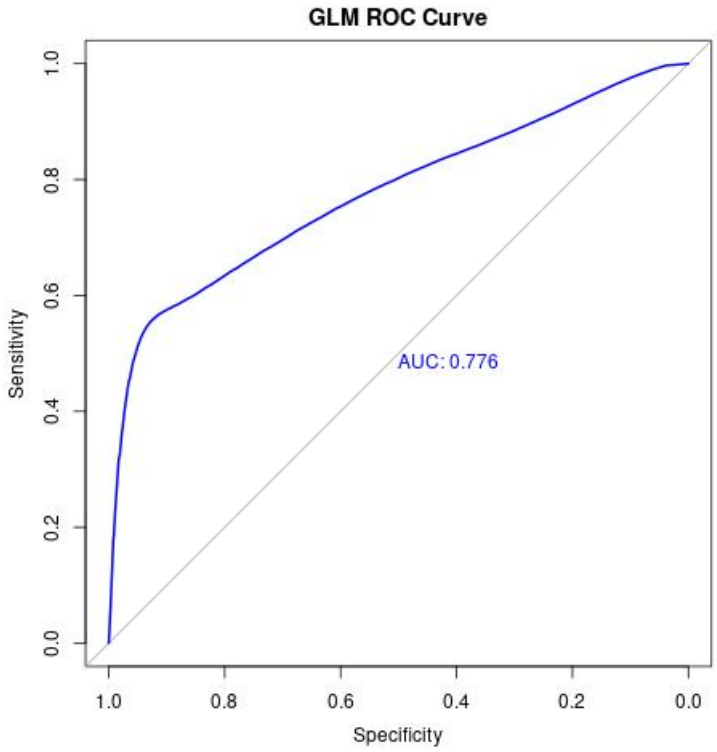
6.1 K Fold Cross Validation

We implement GLM with K-fold cross validation, where k = 10. We compare the model's performance and system performance with parallel and sequential implementation. We also check if GLM performs better with cross validation for our dataset.
We create 10 folds using: cv - createFolds(data$Delay, k=10, list=T).
GLM with cross validation (parallel).

```
Acc:
[1] 0.7
Setting levels: control = 0, case = 1
Setting direction: controls < cases
Area under the curve: 0.7761
   user  system elapsed
 77.290   4.325  10.071
```

GLM with cross validation (sequential).

```
Acc:
[1] 0.7
   user  system elapsed
 61.847   0.591  62.447
```



While the accuracy remains the same, the parallel implementation takes 8 seconds (elapsed time) while the sequential implementation takes over a minute. This is an incredible advantage when working with large datasets. Performance of GLM is the same with and without cross validation.

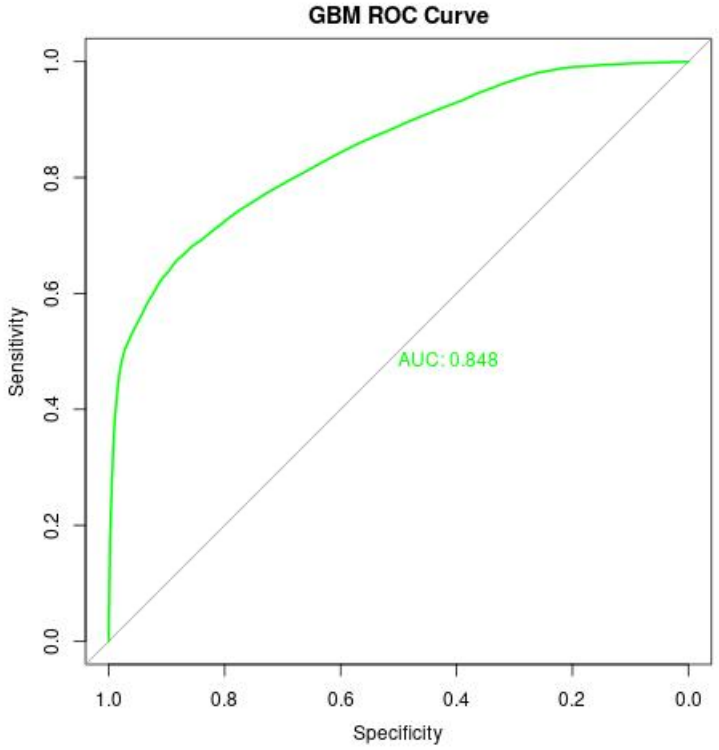
6.2 Boosting

Boosting works by sequentially adding predictors to an ensemble, each new predictor corrects the predecessor. Gradient boosting has the highest test accuracy and AUC out of all the classifiers. Being a sequential algorithm, boosting cannot be implemented in parallel, we implement k-fold cross validation with boosting in parallel, which reduces training time while maintaining the performance. GBM with cross validation (parallel 1500 trees).

```
Acc:
[1] 0.76
Setting levels: control = 0, case = 1
Setting direction: controls < cases
Area under the curve: 0.8479
    user   system  elapsed
5817.049    4.419  583.613
```

GBM with cross validation (sequential 1500 trees).

```
Acc:
[1] 0.76
    user   system  elapsed
4695.444    0.000 4698.581
```

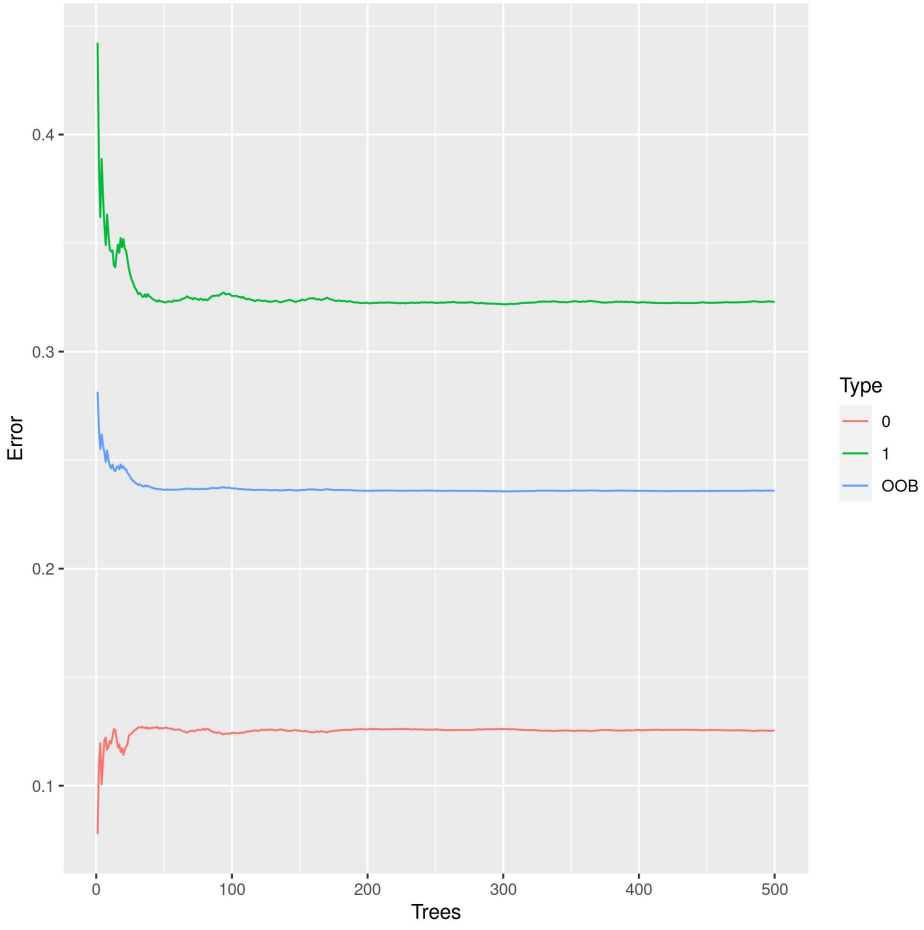


6.3 Random Forest

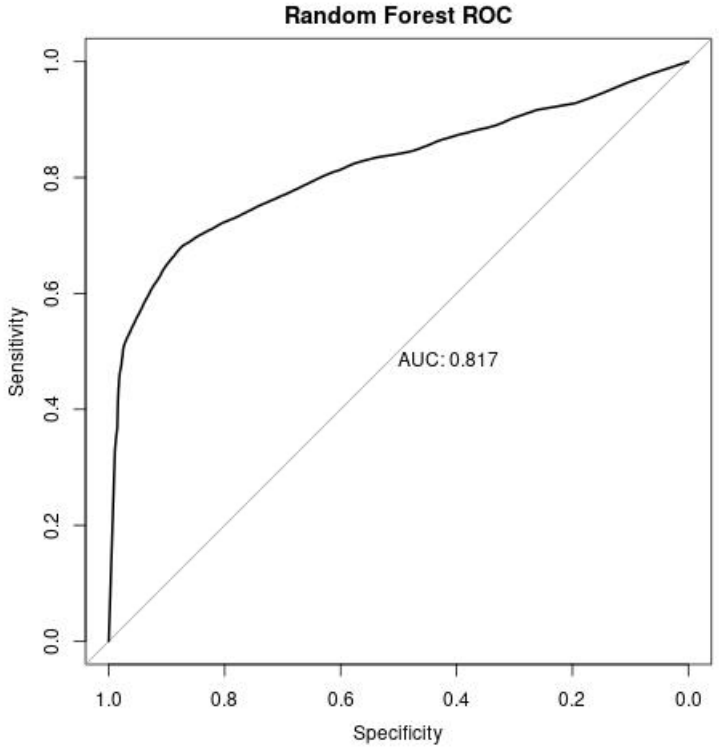
Random forest applies bagging for training. The training dataset is sampled with replacement and uses majority vote to classify a sample. Bagging helps in reduction of variance without increase in the bias.

```
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 23.54%
Confusion matrix:
       0      1 class.error
0 289213  41570   0.1256715
1 135204 284906   0.3218300
```

Out of bag error is calculated on the training set to find optimum number of trees. Out of bag error is the mean error on samples not present in the bootstrapped sample. We observe that the error levels off at round 100 trees for our dataset.



```
Confusion Matrix and Statistics

          Reference
Prediction      0      1
         0 124347  17880
         1  57389 122196

               Accuracy : 0.7661
                 95% CI : (0.7646, 0.7676)
    No Information Rate : 0.5647
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5391

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.8724
            Specificity : 0.6842
         Pos Pred Value : 0.6804
         Neg Pred Value : 0.8743
             Prevalence : 0.4353
         Detection Rate : 0.3797
   Detection Prevalence : 0.5580
      Balanced Accuracy : 0.7783

       'Positive' Class : 1
```



Random forest has test accuracy of 76.6% and AUC of 0.81 which is second

highest after gradient boosting.

6.4 Tabulating Ensemble Method Performance

Method	Sequential Time	Parallel Time	Accuracy	AUC
K-Fold Cross Validation	62.45	10.07	0.70	0.77
Gradient Boosting	4698.58	583.61	0.76	0.85
Random Forest	-	-	0.76	0.82

6.5 Conclusion

- Primary motivation to implement ensemble methods on a large dataset is to implement these methods in parallel to reduce the time take to train without giving up on model performance.
- Parallel implementation of gradient boosting model with k-fold cross validation executes 10x faster than sequential implementation while giving the best performance on the test set.
- GBM and random forest are the best suited models for this dataset. Both have high scores metrics on the test set and low computation time.