PROJECT REPORT: STAT-654

GROUP-13

On

Mobile Price Classification- KAGGLE Dataset

By

Abuzar Patel

UIN: 153005015

TABLE OF CONTENTS

INTRODUCTION

PROBLEM STATEMENT:

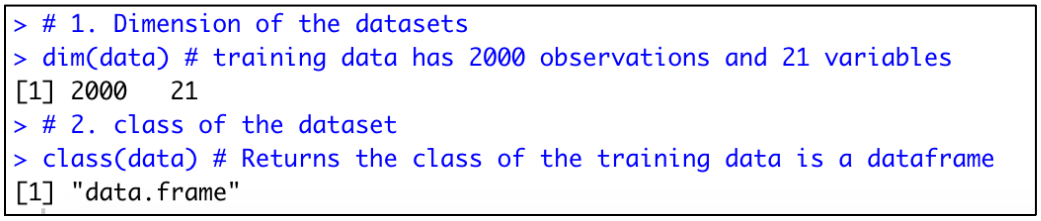
Perform Machine Learning algorithms and build models to classify the price range of the mobiles for the data given in the Kaggle dataset.

The "Mobile Price Classification" dataset was downloaded from Kaggle. The objective of this project is to understand the dataset and build machine learning models to classify the price range of the mobile phones. Mobile prices depend upon several factors such as the Manufacturing costs, geographical location of the market, competitor product price, local taxation rates, profit margin, technical specifications, etc. This dataset consists of only the technical specification of the mobile phone which has a bearing on the price of the mobile phone.

The dataset was studied using the programming language R. The data was visualized using different plots and then various exploratory data analysis (EDA) techniques were performed on the dataset to understand the data. After EDA, four machine learning models were built followed by computing the accuracy rate of each model. Then all the models were compared and the model with the best accuracy rate was selected.

EXPLORATORY DATA ANALYSIS

The dataset was given in the Kaggle in the form of csv file. The file was downloaded in the local computer and read into the R script programming language. The dimension of the dataset gives the size of the data in the form of number of observations and now of predictors. The size of this dataset is 2000* 21. There are 2000 observations in the dataset which is signified by number of rows and 21 variables in the form of columns. The columns are also said to be predictors/features/variables. The class of the dataset is dataframe.



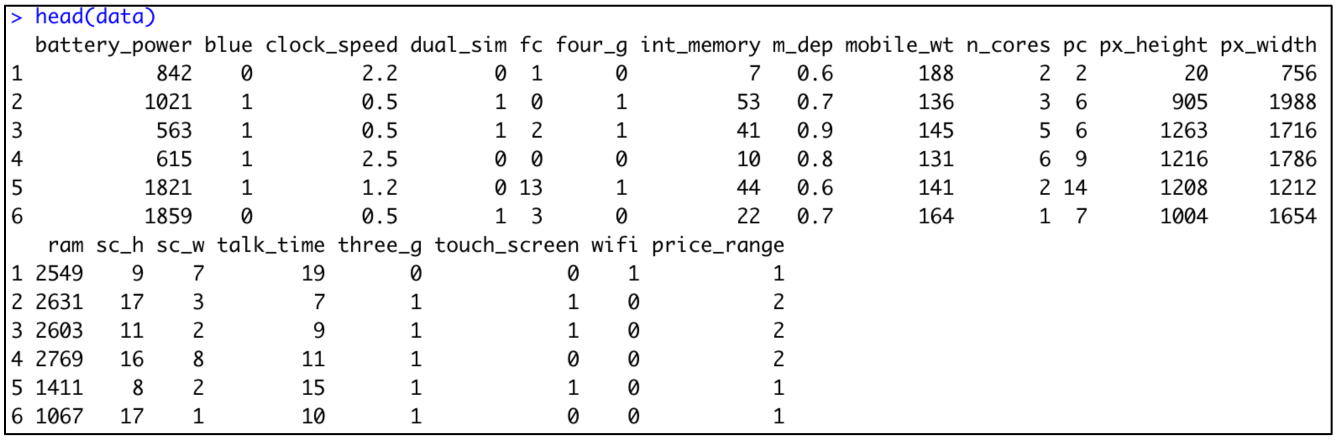
```
> # 1. Dimension of the datasets
> dim(data) # training data has 2000 observations and 21 variables
[1] 2000   21
> # 2. class of the dataset
> class(data) # Returns the class of the training data is a dataframe
[1] "data.frame"
```

The variables/features of the dataset is enlisted as:

battery_power.	: Total energy a battery can store in one time measured in mAh
blu.	: Has bluetooth or not
clock_speed.	: speed at which microprocessor executes instructions
dual_sim.	: Has dual sim support or not
fc	: Front Camera mega pixels
four_g	: Has 4G or not
int_memory.	: Internal Memory in Gigabytes
m_dep.	: Mobile Depth in cm
mobile_wt	: Weight of mobile phone
n_cores	: Number of cores of processor
pc_height	: Primary Camera mega pixels
px_height	: Pixel Resolution Height
px_width.	: Pixel Resolution Width
ram	: Random Access Memory in Mega Bytes
sc_	: Screen Height of mobile in cm

sc_w	**:** Screen Width of mobile in cm
talk_time	**:** longest time that a single battery charge will last when you are talking
three_g	**:** Has 3G or not
touch_screen.	**:** Has touch screen or not
wifi	**:** Has wifi or not
price_range.	**:** This is the target variable with value of 0(low cost), 1(medium cost), 2(high cost) and 3(very high cost).

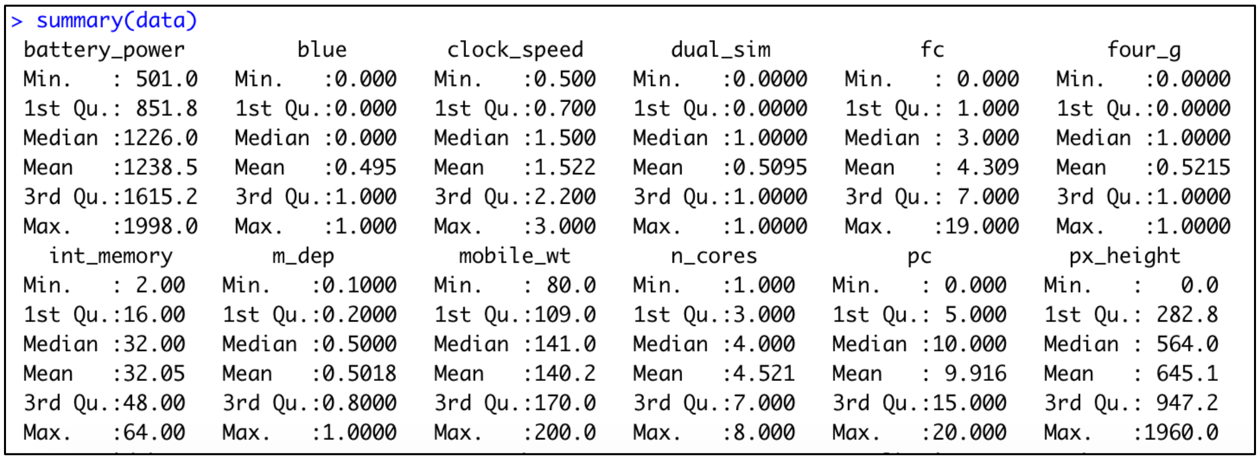
There are 20 predictors and one response variable. The response variable is the price_range which has four classes: 0- low cost , 1: medium cost, 2: high cost, 3: very high cost. Hence the objective of this problem is to classify the price of the mobile in these four classes. The first six observations were read using the head function in R to give more clarity on the values of the dataset and to understand whether the variables are quantitative or qualitative.



```
> head(data)
  battery_power blue clock_speed dual_sim fc four_g int_memory m_dep mobile_wt n_cores pc px_height px_width
1           842    0         2.2        0  1      0          7   0.6       188       2  2        20      756
2          1021    1         0.5        1  0      1         53   0.7       136       3  6       905     1988
3           563    1         0.5        1  2      1         41   0.9       145       5  6      1263     1716
4           615    1         2.5        0  0      0         10   0.8       131       6  9      1216     1786
5          1821    1         1.2        0 13      1         44   0.6       141       2 14      1208     1212
6          1859    0         0.5        1  3      0         22   0.7       164       1  7      1004     1654
   ram sc_h sc_w talk_time three_g touch_screen wifi price_range
1 2549    9    7        19       0            0    1           1
2 2631   17    3         7       1            1    0           2
3 2603   11    2         9       1            1    0           2
4 2769   16    8        11       1            0    0           2
5 1411    8    2        15       1            1    0           1
6 1067   17    1        10       1            0    0           1
```

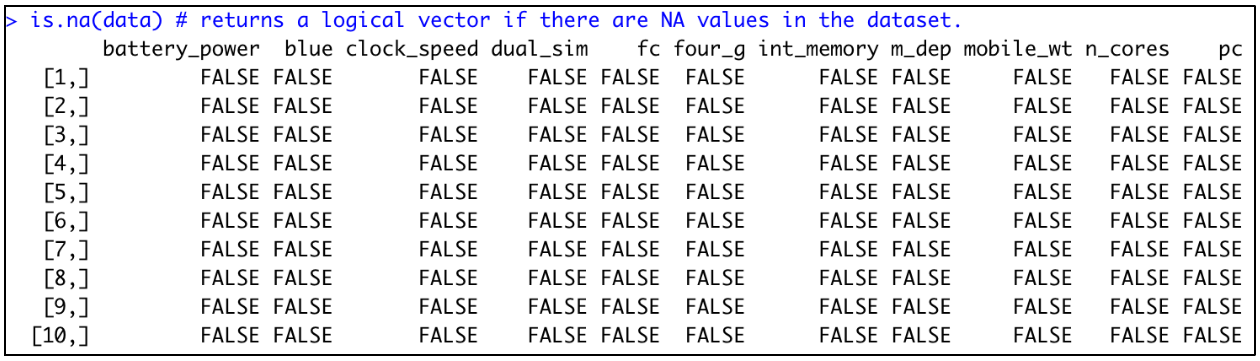
The above six rows tells us that there are six qualitative predictors and fourteen quantitative predictors in the dataset. The qualitative predictors has only two values- 0 and 1 while the quantitative predictors can take any numerical values. Hence we see that the six qualitative predictors are blu, dual_sim, four_g, three_g, touch_screen, wifi while the 14 quantitative predictors are battery_power, clock_speed, fc, int_memory, mobile_wt, n_cores, pc, px_height, px_width, ram, sc_h, sc_w, and talk_time. To know the various statistical parameters of the variables, the summary() was read.

The summary() tells us about the minimum values, 1st quartiles, Median, Mean, 3rd quartile and maximum value of each variable of the dataset.



```
> summary(data)
 battery_power        blue         clock_speed       dual_sim           fc             four_g      
 Min.   : 501.0   Min.   :0.000   Min.   :0.500   Min.   :0.0000   Min.   : 0.000   Min.   :0.0000  
 1st Qu.: 851.8   1st Qu.:0.000   1st Qu.:0.700   1st Qu.:0.0000   1st Qu.: 1.000   1st Qu.:0.0000  
 Median :1226.0   Median :0.000   Median :1.500   Median :1.0000   Median : 3.000   Median :1.0000  
 Mean   :1238.5   Mean   :0.495   Mean   :1.522   Mean   :0.5095   Mean   : 4.309   Mean   :0.5215  
 3rd Qu.:1615.2   3rd Qu.:1.000   3rd Qu.:2.200   3rd Qu.:1.0000   3rd Qu.: 7.000   3rd Qu.:1.0000  
 Max.   :1998.0   Max.   :1.000   Max.   :3.000   Max.   :1.0000   Max.   :19.000   Max.   :1.0000  
   int_memory        m_dep          mobile_wt        n_cores            pc           px_height     
 Min.   : 2.00   Min.   :0.1000   Min.   : 80.0   Min.   :1.000   Min.   : 0.000   Min.   :   0.0  
 1st Qu.:16.00   1st Qu.:0.2000   1st Qu.:109.0   1st Qu.:3.000   1st Qu.: 5.000   1st Qu.: 282.8  
 Median :32.00   Median :0.5000   Median :141.0   Median :4.000   Median :10.000   Median : 564.0  
 Mean   :32.05   Mean   :0.5018   Mean   :140.2   Mean   :4.521   Mean   : 9.916   Mean   : 645.1  
 3rd Qu.:48.00   3rd Qu.:0.8000   3rd Qu.:170.0   3rd Qu.:7.000   3rd Qu.:15.000   3rd Qu.: 947.2  
 Max.   :64.00   Max.   :1.0000   Max.   :200.0   Max.   :8.000   Max.   :20.000   Max.   :1960.0  
```

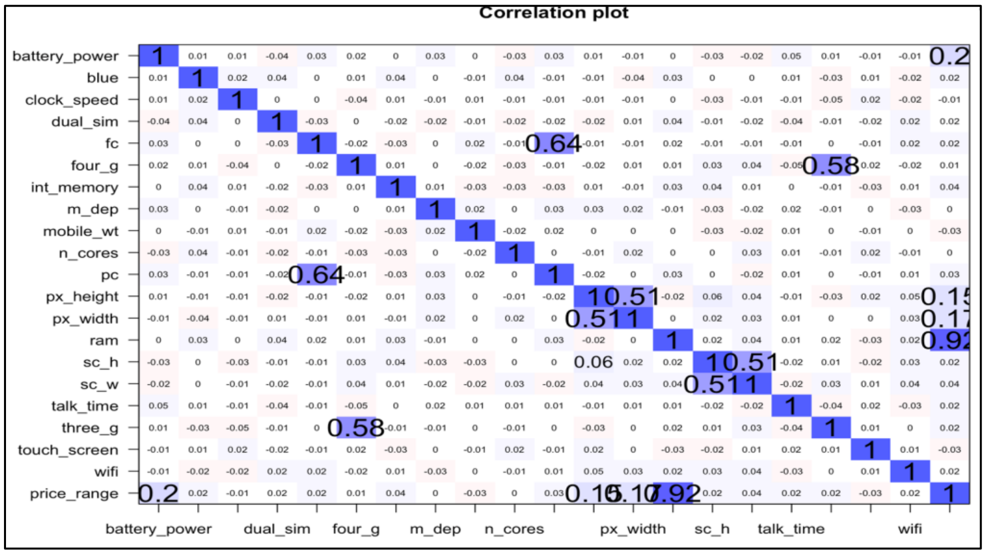
The next step of the EDA was data cleaning. The NA values/ missing values were checked. There was no NA value in the dataset. The output below shows the first ten values of the entire ouput. The FALSE values indicate that there are no NA values in the dataset. Also a sum() was read to calculate the total NA values in the dataset. It indicated a 0 response and hence there was NA value in the dataset.

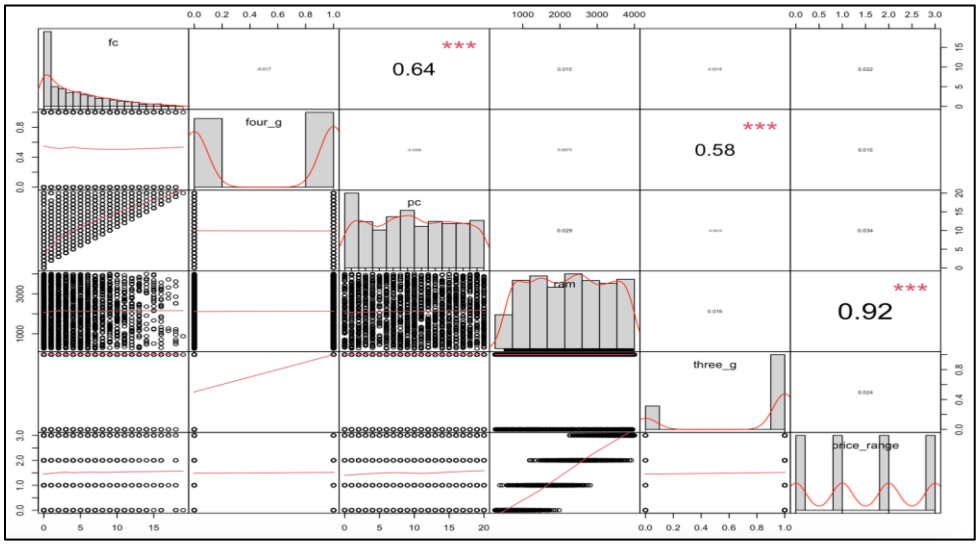


```
> is.na(data) # returns a logical vector if there are NA values in the dataset.
       battery_power  blue clock_speed dual_sim    fc four_g int_memory m_dep mobile_wt n_cores    pc
  [1,]         FALSE FALSE       FALSE    FALSE FALSE  FALSE      FALSE FALSE     FALSE   FALSE FALSE
  [2,]         FALSE FALSE       FALSE    FALSE FALSE  FALSE      FALSE FALSE     FALSE   FALSE FALSE
  [3,]         FALSE FALSE       FALSE    FALSE FALSE  FALSE      FALSE FALSE     FALSE   FALSE FALSE
  [4,]         FALSE FALSE       FALSE    FALSE FALSE  FALSE      FALSE FALSE     FALSE   FALSE FALSE
  [5,]         FALSE FALSE       FALSE    FALSE FALSE  FALSE      FALSE FALSE     FALSE   FALSE FALSE
  [6,]         FALSE FALSE       FALSE    FALSE FALSE  FALSE      FALSE FALSE     FALSE   FALSE FALSE
  [7,]         FALSE FALSE       FALSE    FALSE FALSE  FALSE      FALSE FALSE     FALSE   FALSE FALSE
  [8,]         FALSE FALSE       FALSE    FALSE FALSE  FALSE      FALSE FALSE     FALSE   FALSE FALSE
  [9,]         FALSE FALSE       FALSE    FALSE FALSE  FALSE      FALSE FALSE     FALSE   FALSE FALSE
 [10,]         FALSE FALSE       FALSE    FALSE FALSE  FALSE      FALSE FALSE     FALSE   FALSE FALSE
```

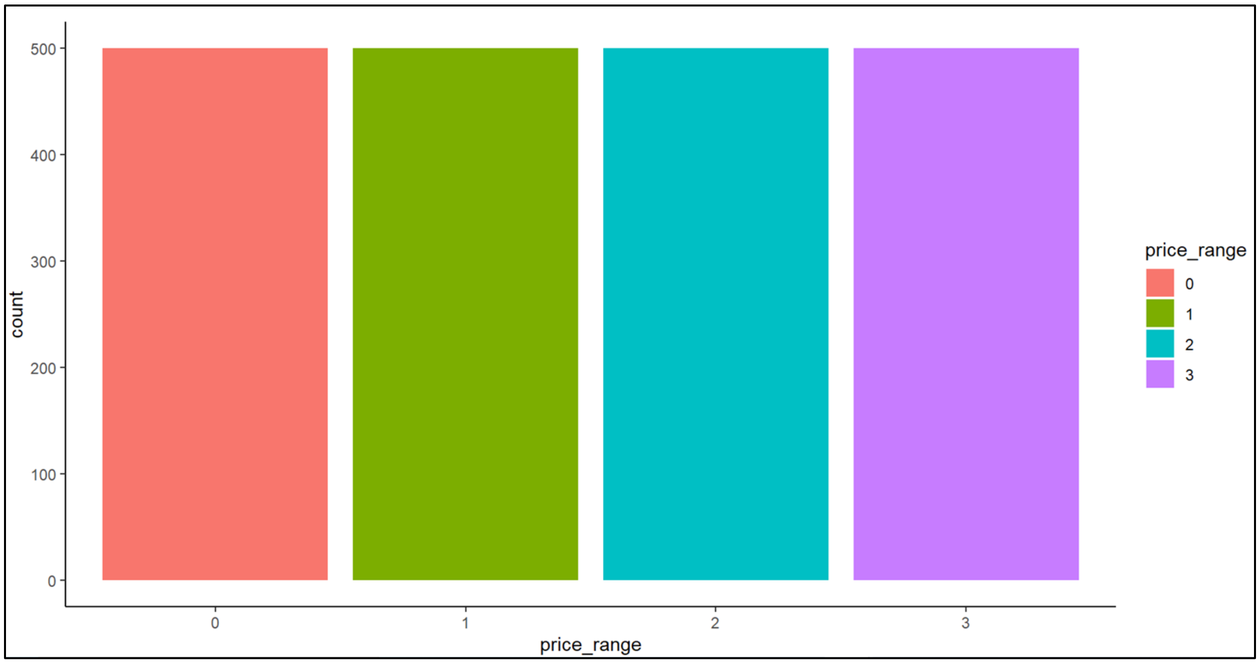
Correlation Plots:

Two correlation plots were plotted to study the correlation of different variables and the response variable. The first plot consists of all the variables while the second plot contains only the variables which have higher correlation coefficients. A variable with high correlation coefficient indicates that it has higher importance in determining the price range of the mobile phone. Hence the variables with high correlation coefficient are front camera, four_g, primary camera, ram and three_g. The variable with highest correlation coefficient is ram having a value of 0.92. The second plot also contains a histogram plot for better clarity.





We plotted the response variable to check if the dataset is balanced or not and the plot below shows that the response of each of the different classes of the price_range of (0,1,2,3) is equal to 500. This equal values of the response variable indicates that it is balanced dataset.

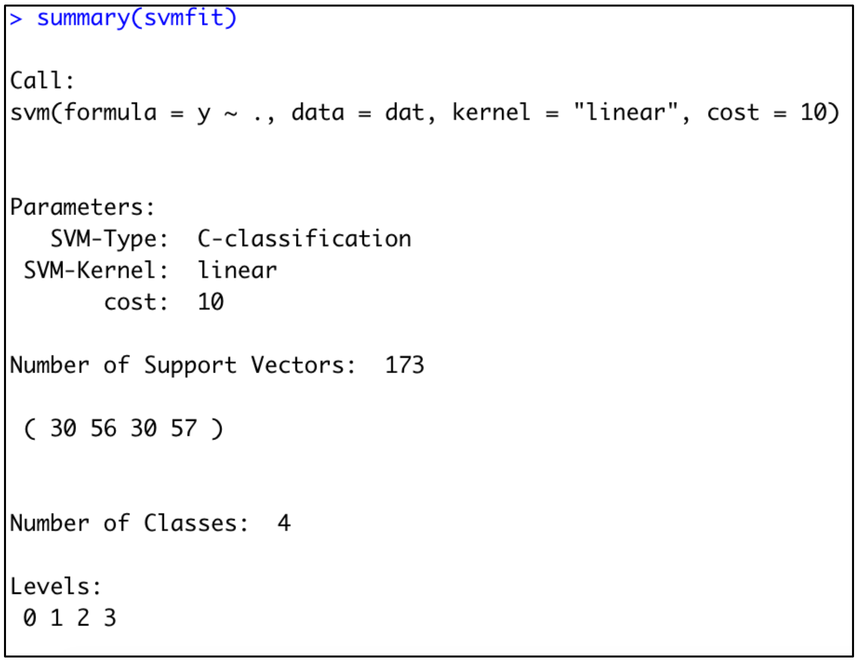


MACHINE LEARNING MODELS

Three different machine learning models were built on the dataset to classify the price range of the mobile phone. The three machine learning algorithms employed on this dataset were Support Vector machines- Linear kernel and Radial kernel, Random Forests and Multinomial Logistic Regression. Training and test errors were calculated after building each model. The three models were then compared at the end to select the model with the highest test accuracy.

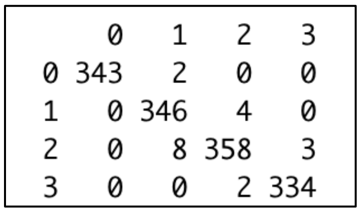
1. Support Vector Machines – (LINEAR kernel):

It works on the principle of making hyperplanes which is used for the separation of the observations into different classes. There is a cost penalty parameter called kernel which defines the margin from the hyperplane and helps to find the support vectors. The support vectors are the observations which lies or within that margin. Before building the machine learning model, the data was split into training data and test data in the ratio of 70:30. The model was built on 70% of the observations and validated on the remaining 30% of the test data.

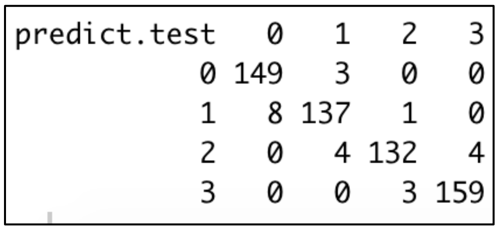


```
> summary(svmfit)

Call:
svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10

Number of Support Vectors:  173

 ( 30 56 30 57 )


Number of Classes:  4

Levels:
 0 1 2 3
```

After fitting the support vector machine with linear kernel, it was observed that it has 173 support vectors with the cost parameter equal to ten. The various classification values into different classes of 0,1,2,3 are 30,56,30,57. A confusion matrix was created for both training and test datasets. The confusion matrix of both training and test sets are shown below.



```
    0   1   2   3
0 343   2   0   0
1   0 346   4   0
2   0   8 358   3
3   0   0   2 334
```

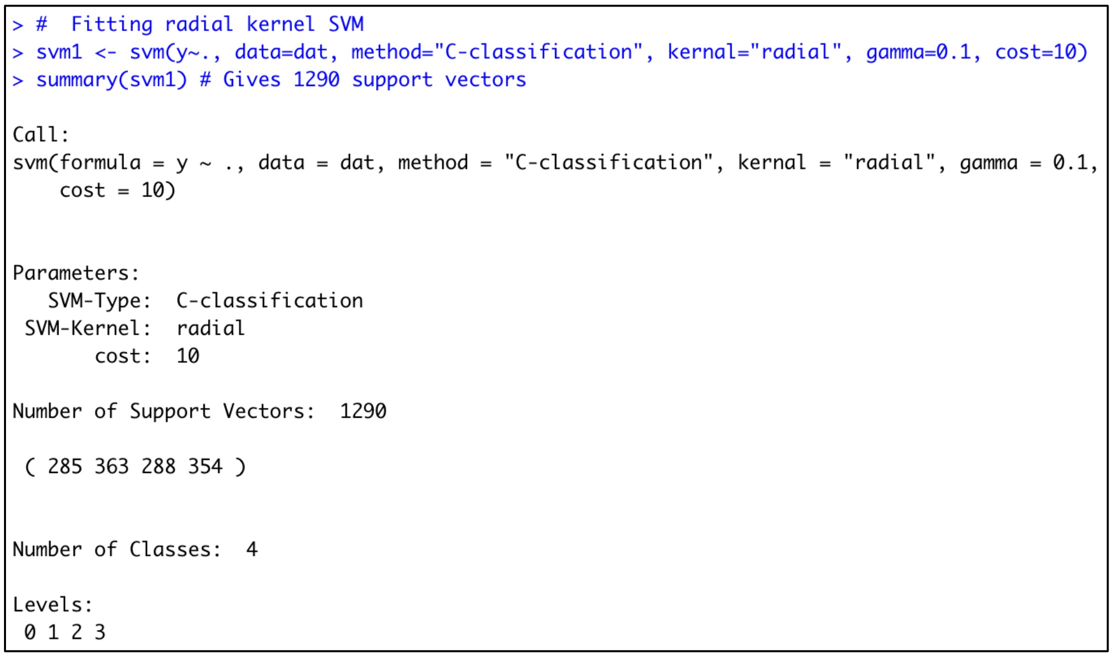


```
predict.test   0   1   2   3
           0 149   3   0   0
           1   8 137   1   0
           2   0   4 132   4
           3   0   0   3 159
```

The training error is 0.95% and test error is 3.87 %.

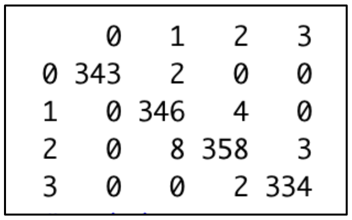
- **. Support Vector Machines – (Radial kernel):**

The Support Vector Machine with radial kernel was fit and confusion matrix created.

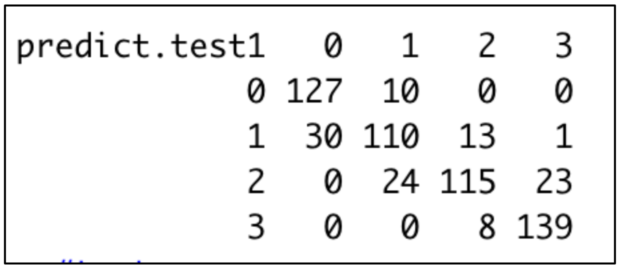


```
> #  Fitting radial kernel SVM
> svm1 <- svm(y~., data=dat, method="C-classification", kernal="radial", gamma=0.1, cost=10)
> summary(svm1) # Gives 1290 support vectors

Call:
svm(formula = y ~ ., data = dat, method = "C-classification", kernal = "radial", gamma = 0.1,
    cost = 10)


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

Number of Support Vectors:  1290

 ( 285 363 288 354 )


Number of Classes:  4

Levels:
 0 1 2 3
```

The output of the support vector machine with radial kernel indicates that there are 1290 support vectors for 4 classes. Similarly, confusion matrix was created for training and test dataset. This gives the training error to be 3.83% and the test error to be 18.03 %



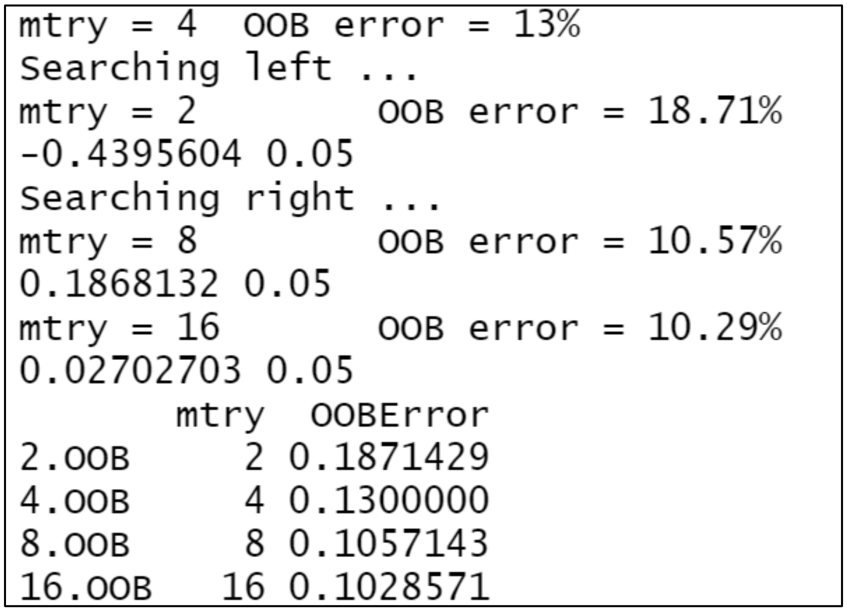
```
    0   1   2   3
0 343   2   0   0
1   0 346   4   0
2   0   8 358   3
3   0   0   2 334
```



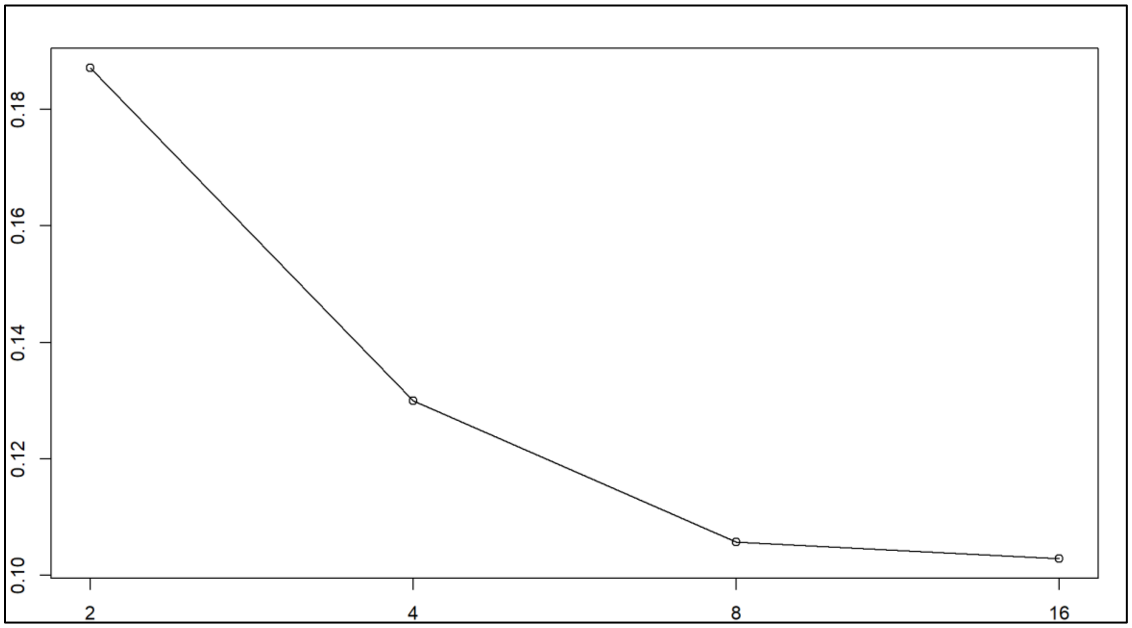
```
predict.test1   0   1   2   3
            0 127  10   0   0
            1  30 110  13   1
            2   0  24 115  23
            3   0   0   8 139
```

2. Random Forest Model.

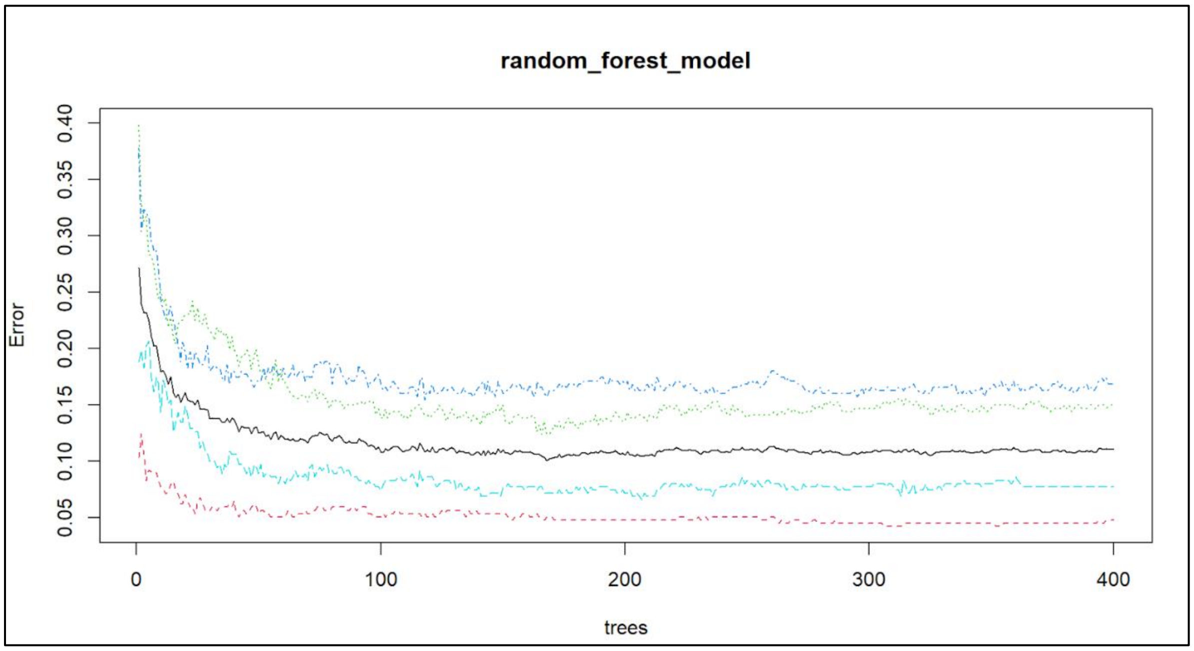
The random forest method is a special case of bagging where instead of using all the variables at random it uses selected number of variables at random. Due to this reason and it's out of bag sampling random forest is not subjected to overfitting, however one of the cons of random forest is that if the future data is not representative of training data, then the method can lead to poor results. In our case both training data and testing data are representative. After conducting a 70-30 split on the entire data and using a 10-fold CV Random Forest was conducted. We initially ran Cross validation to tune the random forest hyperparameter namely "mtry" which is the number of variables use for splitting the trees. The table below summarizes the CV result against OOB error and determine lowest error at 16 variables. However, the OOB result for 8 is very similar to 16 and because the reduction from 8 to 16 value of mtry does not give a significant reduction in OOB we decided to go with 8 mtry. The graph clearly shows this and our reasoning behind selected 8 was motivated by computational capabilities. The x axis is number of variables whereas y is OOB.



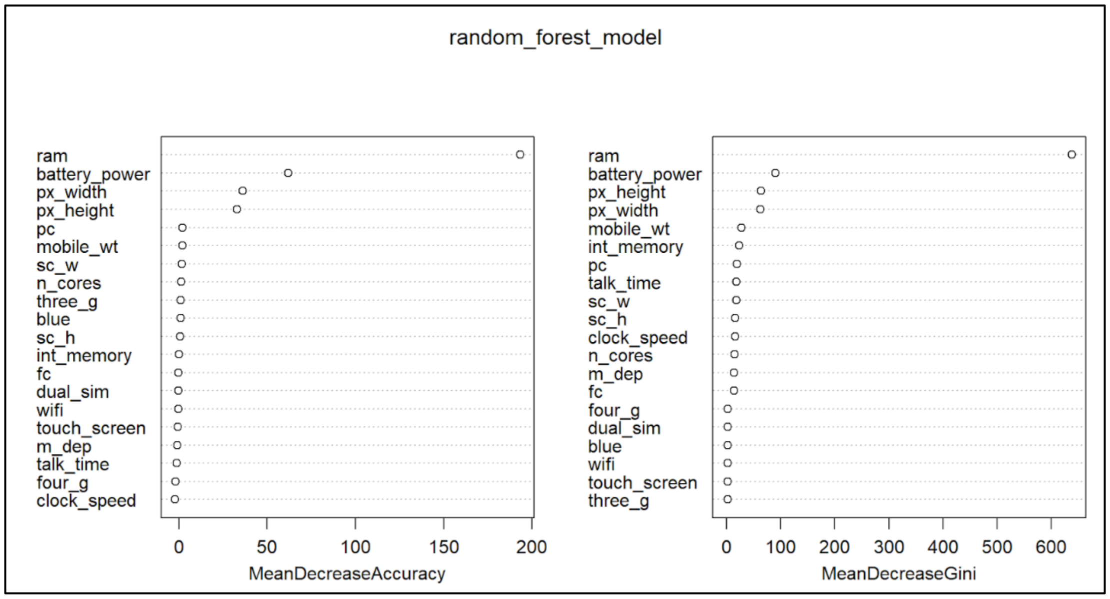
```
mtry = 4  OOB error = 13%
Searching left ...
mtry = 2        OOB error = 18.71%
-0.4395604 0.05
Searching right ...
mtry = 8        OOB error = 10.57%
0.1868132 0.05
mtry = 16       OOB error = 10.29%
0.02702703 0.05
       mtry  OOBError
2.OOB     2 0.1871429
4.OOB     4 0.1300000
8.OOB     8 0.1057143
16.OOB   16 0.1028571
```



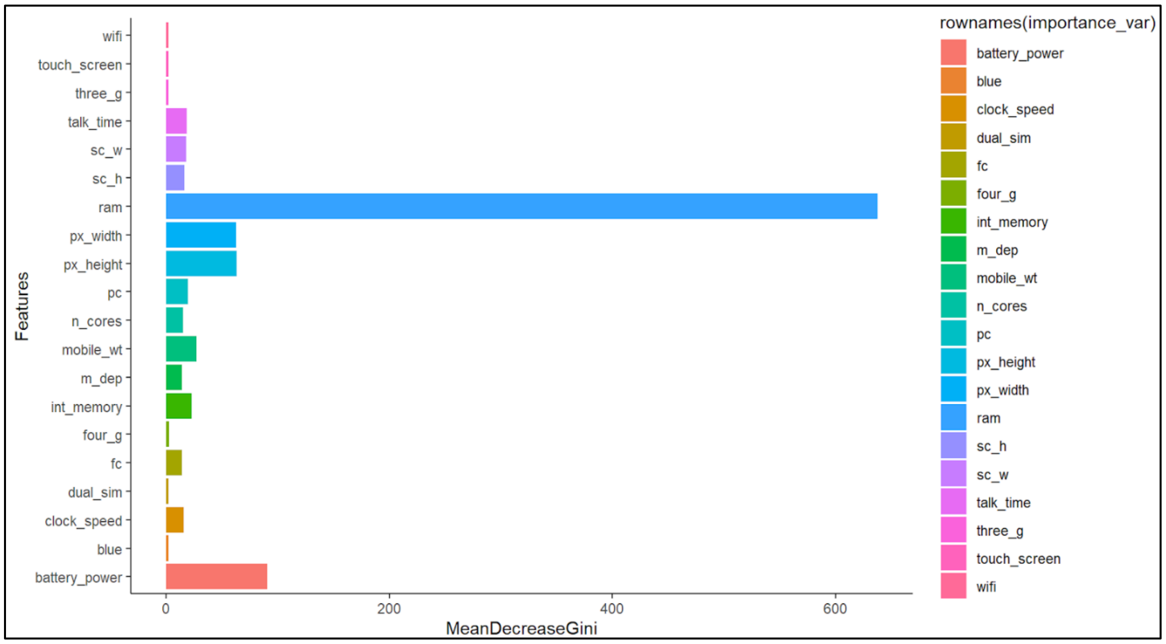
After selecting an optimal mtry value next job is to determine the number of trees required. This number by default in R is set to be 500 and our desire is to select an optimal number of trees which will stabilize Out of Bag (OOB) error. A similar method such as ntry was used to determine ntree and we came up with 400 number of trees beyond which increasing the number of trees had little or no effect on the Out of Bag Error. This can be verified by the plot below. Here different pots represent error rate for 4 different classes and one overall OOB error. At this point we don’t really care which line belongs to which class but rather if the errors have stabilized and the indeed have.



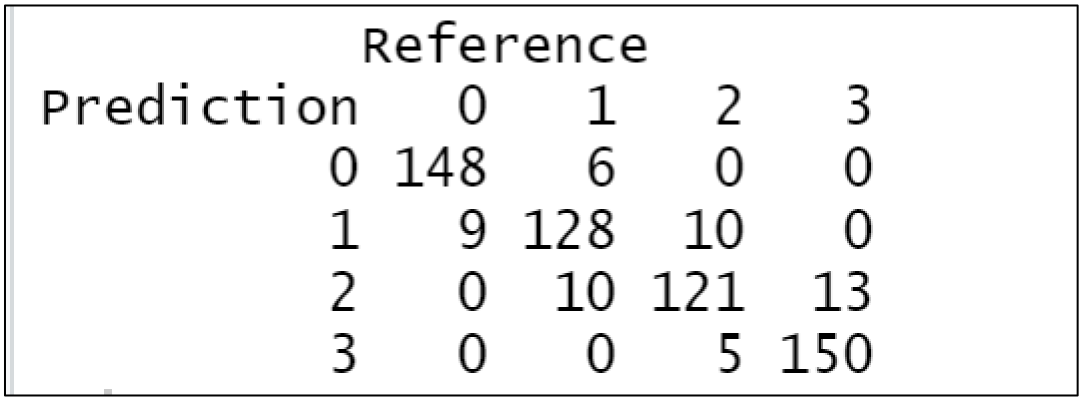
The randomForest package also allows us to look at the Importance of Variables by considering the mean Decrease Accuracy and MeanDecreaseGini (a measure of how pure the node Is). The Mean decrease accuracy can be thought of how much of the accuracy is contributed by that one variable and therefore the higher the value the better it is. Below the two plots ranks the variable and higher the value the important the variable. We see that both the MeanDecrease Gini and Mean Decrease Accuracy nearly gives us the same variable as important.



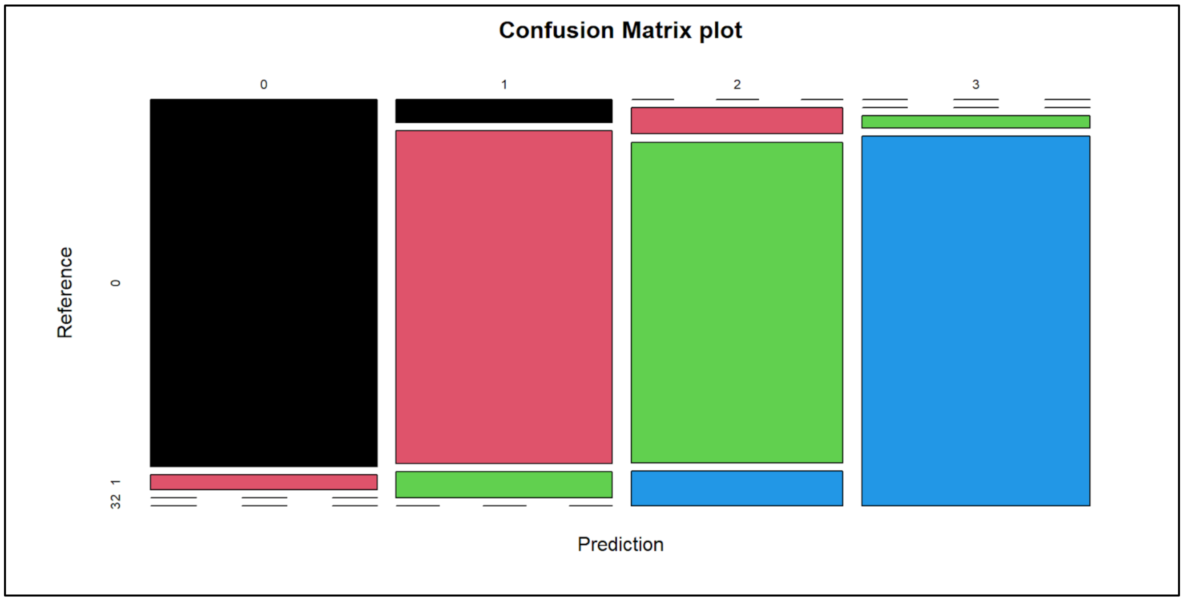
The barplot below represents the same result,



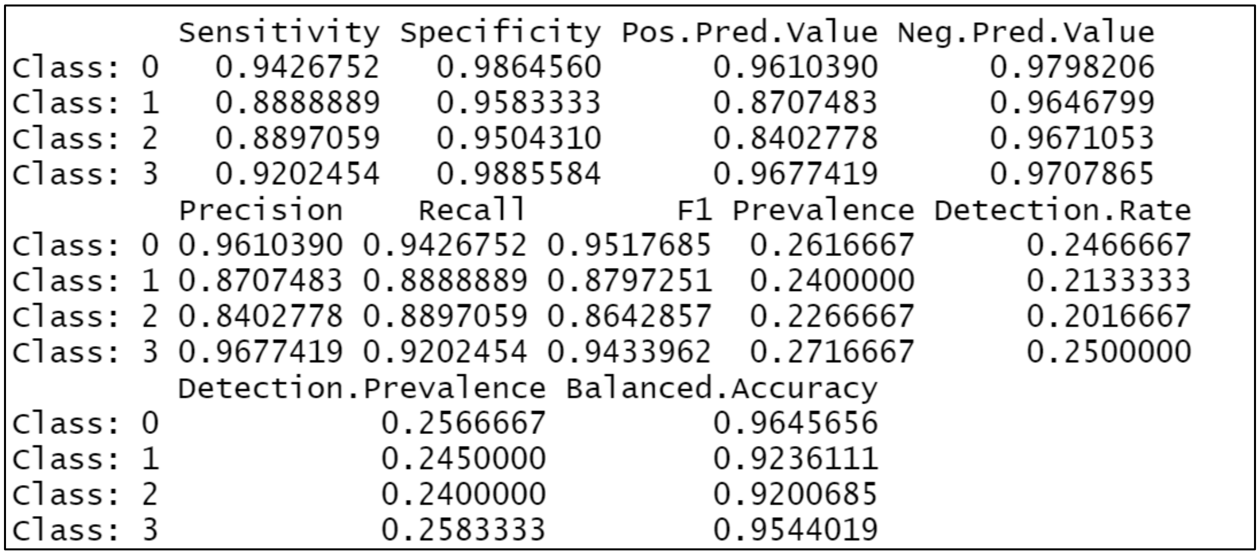
After training and considering important variable, we then predict our model on the testing data to estimate $y^{\wedge}\ values$. The predicted confusion matrix on the testing data set is given below along with a visual representation of Confusion Matrix.



```
               Reference
Prediction    0   1   2   3
         0  148   6   0   0
         1    9 128  10   0
         2    0  10 121  13
         3    0   0   5 150
```

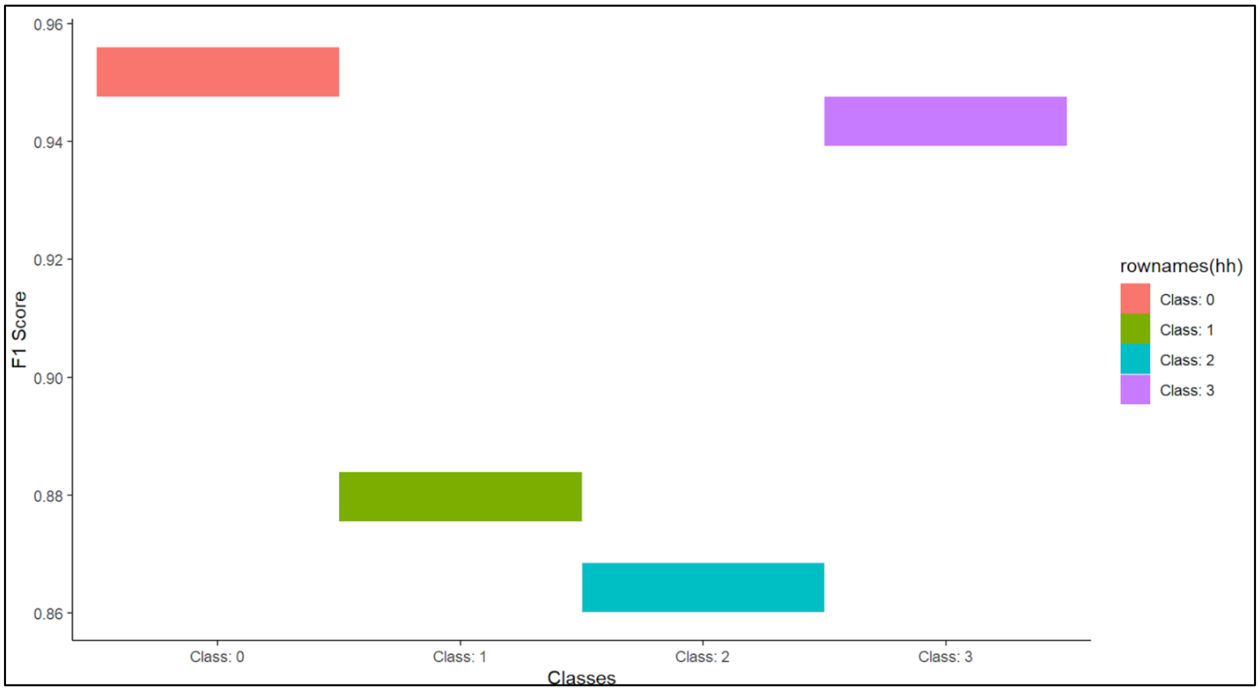


As we can see that our diagonal values on the plot and the table are predicted accurately almost all the time and dominates the confusion table. We then see how well our performance of individual classes is by looking at their respective F1 score. F1 score is a combination of precision and recall getting a robust estimate of accuracy. The graph below displays the F1 score for each classes along with the overall accuracy on the final validation set as 91.67



```
         Sensitivity Specificity Pos.Pred.Value Neg.Pred.Value
Class: 0   0.9426752   0.9864560      0.9610390      0.9798206
Class: 1   0.8888889   0.9583333      0.8707483      0.9646799
Class: 2   0.8897059   0.9504310      0.8402778      0.9671053
Class: 3   0.9202454   0.9885584      0.9677419      0.9707865
         Precision    Recall        F1 Prevalence Detection.Rate
Class: 0 0.9610390 0.9426752 0.9517685  0.2616667      0.2466667
Class: 1 0.8707483 0.8888889 0.8797251  0.2400000      0.2133333
Class: 2 0.8402778 0.8897059 0.8642857  0.2266667      0.2016667
Class: 3 0.9677419 0.9202454 0.9433962  0.2716667      0.2500000
         Detection.Prevalence Balanced.Accuracy
Class: 0            0.2566667         0.9645656
Class: 1            0.2450000         0.9236111
Class: 2            0.2400000         0.9200685
Class: 3            0.2583333         0.9544019
```

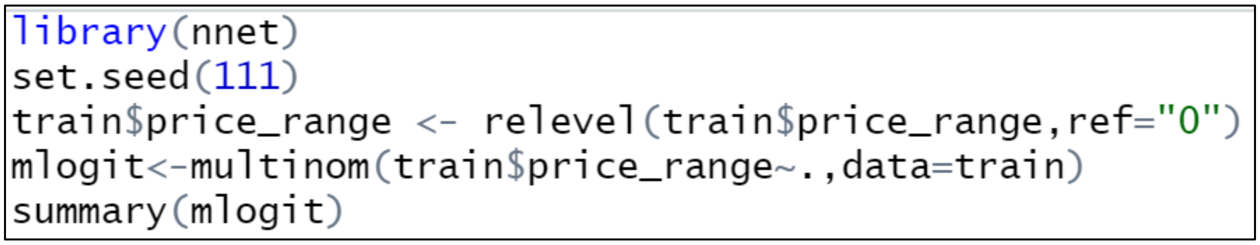
Below is a visual representation of Individual F1 Score. Lastly the final accuracy f our model was 91.67 percent.



3. MultiNomial Logistic Regression

The next model ran is the Multinomial Logistic Regression (MLR) model, because our response consists of more than two classes this is a perfect example of MLR. In this we began by setting baseline class, this class can be any of your class (0 in our case) and every result after setting class 1 should be compared to the baseline.

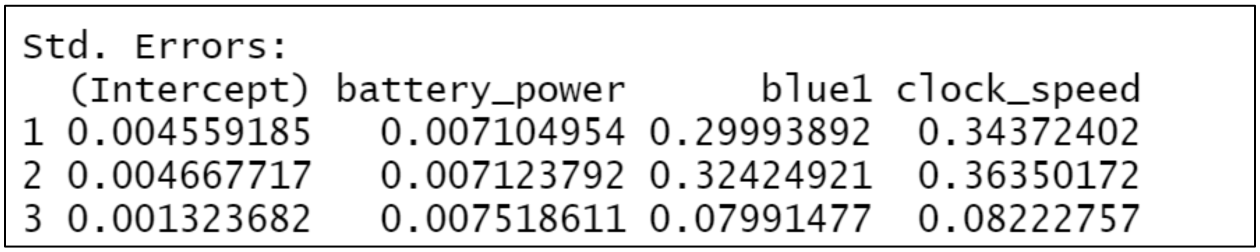
Below is the code used to run MLR:



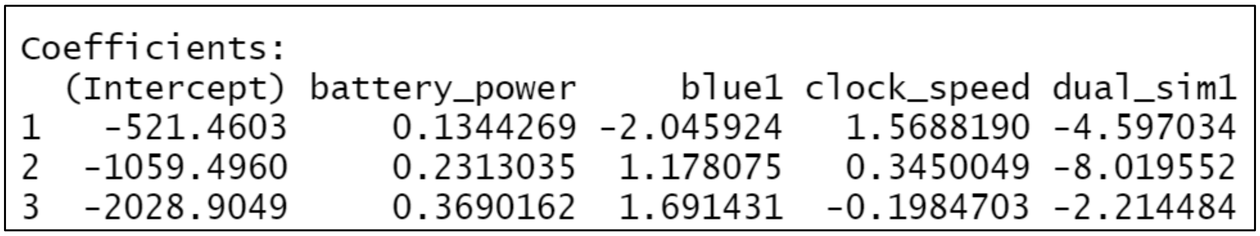
```
library(nnet)
set.seed(111)
train$price_range <- relevel(train$price_range,ref="0")
mlogit<-multinom(train$price_range~.,data=train)
summary(mlogit)
```

A snippet of summary output is show below and the output from MLR seems to differ from that of simple LR in the sense that there are no P vales or Z values associated with the variables. The result from summary consists of only the standard errors and coefficient.

Interpreting the results of MLR:

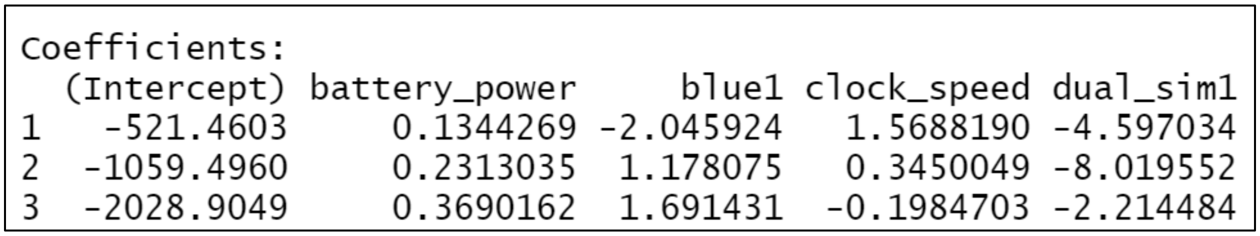


```
Std. Errors:
  (Intercept) battery_power      blue1 clock_speed
1 0.004559185   0.007104954 0.29993892  0.34372402
2 0.004667717   0.007123792 0.32424921  0.36350172
3 0.001323682   0.007518611 0.07991477  0.08222757
```



```
Coefficients:
  (Intercept) battery_power     blue1 clock_speed dual_sim1
1   -521.4603     0.1344269 -2.045924   1.5688190 -4.597034
2  -1059.4960     0.2313035  1.178075   0.3450049 -8.019552
3  -2028.9049     0.3690162  1.691431  -0.1984703 -2.214484
```

As said in the snippet of summary above we can see the Coefficients and Std. Errors. Lets for a moment consider the Coefficient only.



```
Coefficients:
  (Intercept) battery_power     blue1 clock_speed dual_sim1
1   -521.4603     0.1344269 -2.045924   1.5688190 -4.597034
2  -1059.4960     0.2313035  1.178075   0.3450049 -8.019552
3  -2028.9049     0.3690162  1.691431  -0.1984703 -2.214484
```

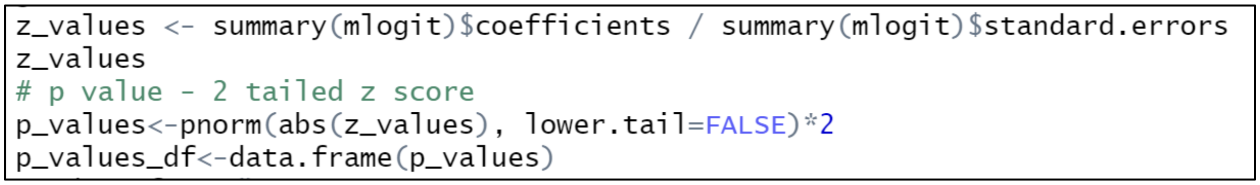
Below is the Mathematical relationship with the log odds of response with reference to baseline.

$$\log\left(\frac{p(Price = 1)}{p(Price = 0)}\right) = b10 + b_{11}X_1 + b_{12}X2 + b_{13} * X_3 \ldots \ldots$$

The 1, 2, 3 in the coefficient table represent class name and if we look at the first row or class 1 in the Coefficient table then the numerator in the above example will take the value of 1, if considering 2 (not shown here) the numerator will take the value of 2 and similarly for 3. Considering only the row 1 or class 1 for the moment, we can see that our B1 for battery_power (X1) is 0.1344269. Firstly, we can see that there is a positive relationship with battery_hour and the log odds of a relationship of a mobile phone to be classified as 1 in comparison to 0. We can conclude that with a one unit increase in battery_power the log odds increase by 0.1344. and similarly for rest of the lasses and variables.

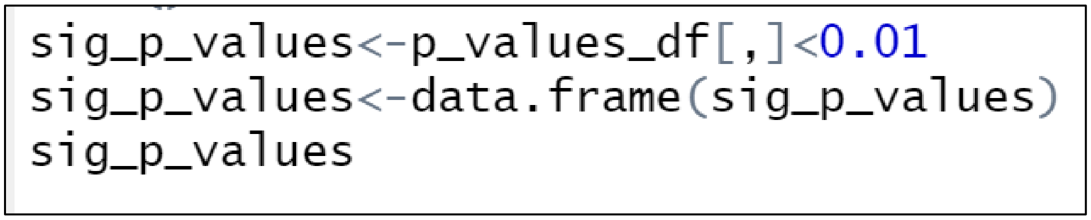
How to Check for Significance:

In normal LR the summary output consists of Z value and P value however in MLR the output consists of only the std errors and the coefficient and therefore we compute the z value and p value manually. Z value is simply the sum of coefficient divided by sum of std errors. Next is the computation of P values. The P value is just 2 tailed z value in a normal distribution hence we compute the p value using the following code.



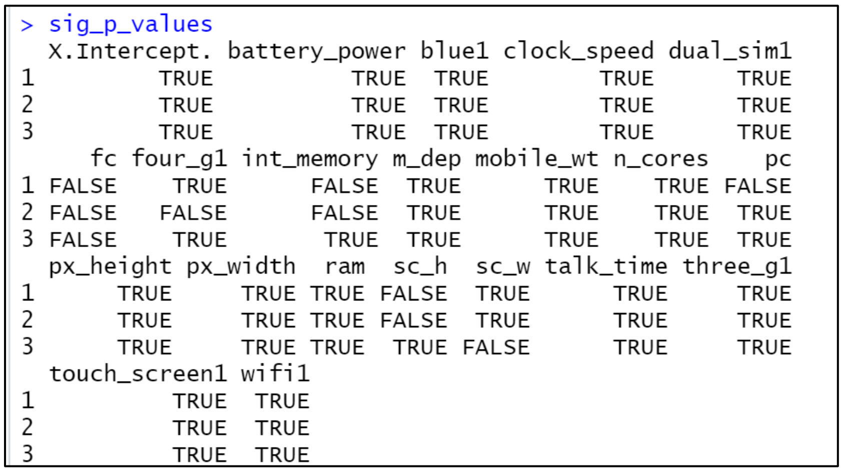
```
z_values <- summary(mlogit)$coefficients / summary(mlogit)$standard.errors
z_values
# p value - 2 tailed z score
p_values<-pnorm(abs(z_values), lower.tail=FALSE)*2
p_values_df<-data.frame(p_values)
```

At this time we are not interested in the actual p values but rather if the variables are significant or not and we do this by setting an alpha value at 0.01 and filtering for all the variables with a p value less than alpha value.



```
sig_p_values<-p_values_df[,]<0.01
sig_p_values<-data.frame(sig_p_values)
sig_p_values
```

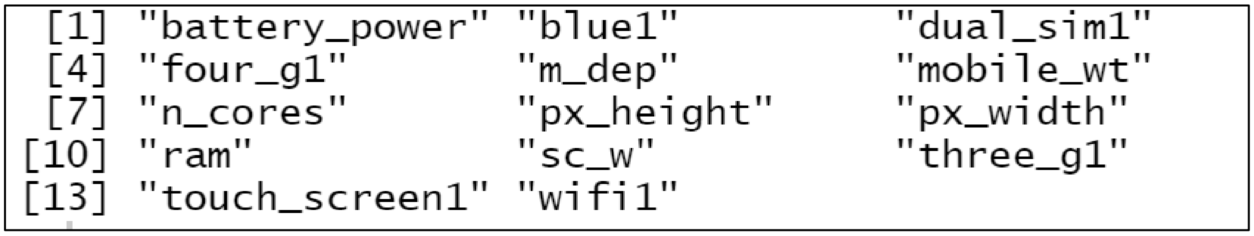
This results in the logical matrix given below



```
> sig_p_values
  X.Intercept. battery_power blue1 clock_speed dual_sim1
1         TRUE          TRUE  TRUE        TRUE      TRUE
2         TRUE          TRUE  TRUE        TRUE      TRUE
3         TRUE          TRUE  TRUE        TRUE      TRUE
     fc four_g1 int_memory m_dep mobile_wt n_cores    pc
1 FALSE    TRUE      FALSE  TRUE      TRUE    TRUE FALSE
2 FALSE   FALSE      FALSE  TRUE      TRUE    TRUE  TRUE
3 FALSE    TRUE       TRUE  TRUE      TRUE    TRUE  TRUE
  px_height px_width  ram  sc_h  sc_w talk_time three_g1
1      TRUE     TRUE TRUE FALSE  TRUE      TRUE     TRUE
2      TRUE     TRUE TRUE FALSE  TRUE      TRUE     TRUE
3      TRUE     TRUE TRUE  TRUE FALSE      TRUE     TRUE
  touch_screen1 wifi1
1          TRUE  TRUE
2          TRUE  TRUE
3          TRUE  TRUE
```

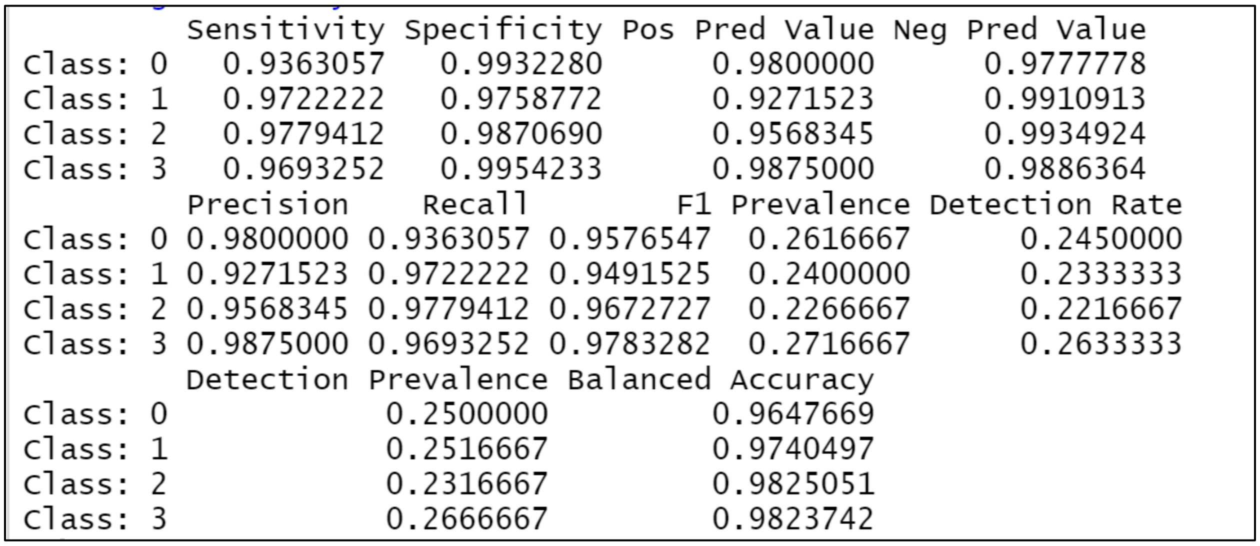
Lastly, to determine which variables are significant we select all the variables that has a value of TRUE (significant) in all three classes with reference to baseline and determine those as significant variables.

Below is the list of variable names deemed to be significant through MLR. These variables are like the ones determine by Random Forest.

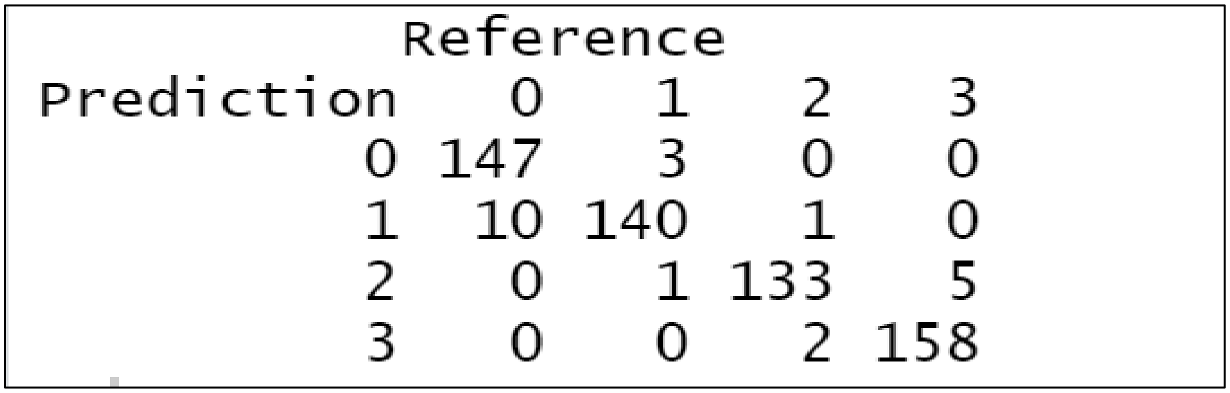


```
 [1] "battery_power" "blue1"         "dual_sim1"
 [4] "four_g1"       "m_dep"         "mobile_wt"
 [7] "n_cores"       "px_height"     "px_width"
[10] "ram"           "sc_w"          "three_g1"
[13] "touch_screen1" "wifi1"
```

Below is the result from confusion matrix for MLR along with the confusion matrix



```
         Sensitivity Specificity Pos Pred Value Neg Pred Value
Class: 0   0.9363057   0.9932280      0.9800000      0.9777778
Class: 1   0.9722222   0.9758772      0.9271523      0.9910913
Class: 2   0.9779412   0.9870690      0.9568345      0.9934924
Class: 3   0.9693252   0.9954233      0.9875000      0.9886364
         Precision    Recall        F1 Prevalence Detection Rate
Class: 0 0.9800000 0.9363057 0.9576547  0.2616667      0.2450000
Class: 1 0.9271523 0.9722222 0.9491525  0.2400000      0.2333333
Class: 2 0.9568345 0.9779412 0.9672727  0.2266667      0.2216667
Class: 3 0.9875000 0.9693252 0.9783282  0.2716667      0.2633333
         Detection Prevalence Balanced Accuracy
Class: 0            0.2500000         0.9647669
Class: 1            0.2516667         0.9740497
Class: 2            0.2316667         0.9825051
Class: 3            0.2666667         0.9823742
```



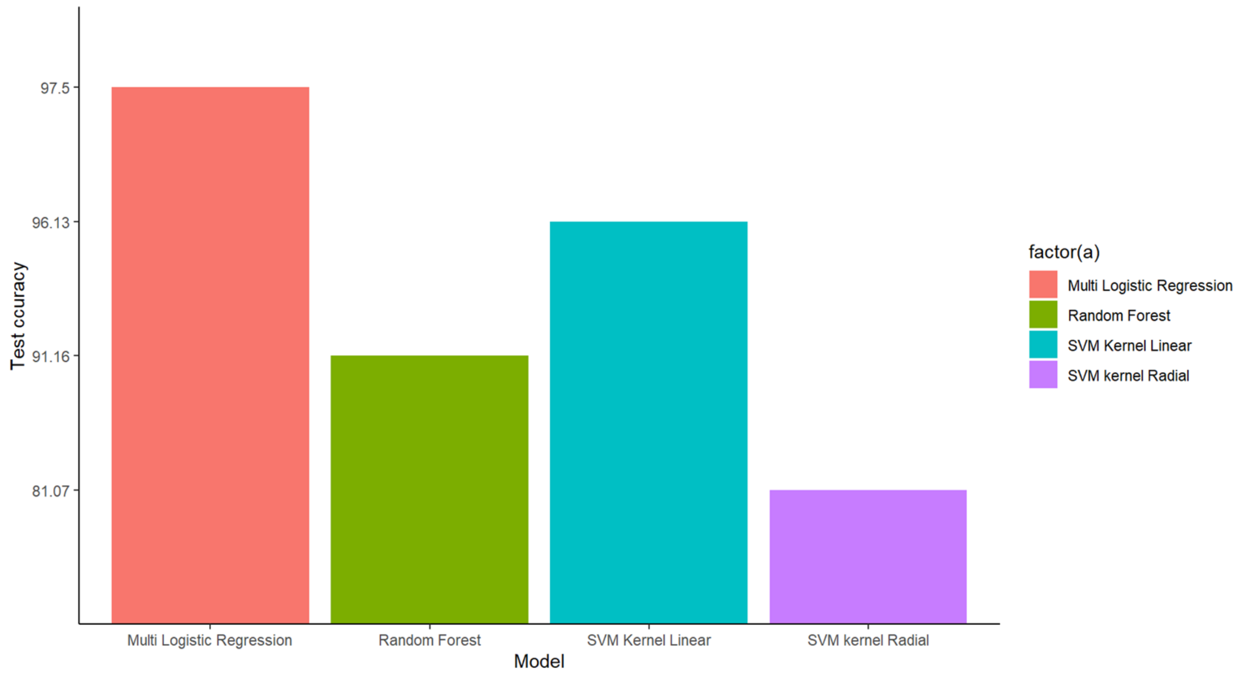
```
          Reference
Prediction   0   1   2   3
         0 147   3   0   0
         1  10 140   1   0
         2   0   1 133   5
         3   0   0   2 158
```

The overall testing accuracy for our MLR came out to be the highest at 97.5

COMPARISON OF MACHINE LEARNING MODELS

On comparing the test accuracy of the four machine learning models which were built on the dataset, it was found that Multinomial Logistic Regression has the best test accuracy of 97.5% followed by Support Vector Machine using Linear kernel of 96.13% while the Random Forest has the test accuracy of 91.16% and the Support Vector Machine with Radial kernel has lowest test accuracy of 81.07%.

The test accuracy has been shown in the bar chart below:



CONCLUSION, LIMITATIONS, FUTURE SCOPE

Mobile pricing is a very complex process, and it depends upon several factors other than the technical specifications. This dataset only consisted of the technical specifications of the mobile phones and the machine learning models were built to classify the **price_range** of the mobile phones based on 2000 observations. The test accuracy for the Multinomial Logistic Regression was the best with accuracy rate of 97.5% while the Support Vector Machine with test accuracy of 81% performed the worst of the four models.

The limitation of the dataset was that it consisted of only the technical specifications. Also, there can be many other technical specifications which has not been considered here to determine the price_range of the mobile phone such as graphics, sound quality, if the memory size can be increased or not etc. Hence if more additional technical features are considered, then the model can perform differently.

The future scope of the project can be if the price range can be defined in terms of quantitative values instead of only four different classes, then different machine learning models can be built to predict the price of the mobile phones and then it can be classified into different class.