# Mini – Project

# Report on Supervised Machine Learning

Report By

Chakradhar Varma

# Table of Contents

# 1. Project Objective

The objective of the project is to perform supervised machine learning techniques on the given data, which is from a personal loans campaign executed by MyBank. The Classification Models such as Classification Tree, Random Forest and Neural Network model are to be built on the data-set in order to predict the response of the new customers. Model Performance is to be measured on the Development sample and should be validated on the Holdout sample to ensure the model is not overfit. And finally performance of the three models needs to be compared and the insights should be provided.

# 2. Solutions

This Solutions section will explain each part of the project in the following steps:

1. Preliminary and Exploratory Data Analysis.
2. Classification Tree Model.
3. Random Forest Model.
4. Neural Network Model.
5. Model Comparison.

The Source code for all the above steps is attached in the Appendix A Section.

## 2.1 Preliminary and Exploratory Data Analysis

The data provided is from a Personal Loans Campaign executed by MyBank. 20000 customers were targeted with an offer of Personal Loans at 10% interest rate. 2512 customers out of 20000 responded expressing their need for Personal Loan.

In the given data set there are total of 20000 observations with 40 variables of interest. It has features of different classes like integer, numeric and Factor types. These features are basically providing the demographic, behavioral and transaction information of the customers. The data has been tested for any missing values and found out that there is no missing values in the sample.

As part of the Exploratory Data Analysis, each variable is summarized and descriptive statistics is performed to check the normality. Initially Filter Method is applied as a feature selection technique to see if any feature can be eliminated. The features that are found to be in-significant are removed from the data set and the remaining variables are validated for selection using wrapper method i.e., Boruta Algorithm.

It is observed that the percentage of non-responders (87.44%) is way more than the responders. This implies that, data might need to be balanced in order to achieve good classification results.

## 2.2 Classification Tree Model (CART)

Decision tree is a type of supervised learning algorithm that can be used in both regression and classification problems. It works for both categorical and continuous input and output variables. A Classification tree is a technique used to predict qualitative response against both categorical and continuous input variables.

A Classification tree is built through a process known as binary recursive partitioning. This is an iterative process of splitting the data into partitions, and then splitting it up further on each of the branches. In this section the model building of classification tree is explained in the following multiple steps.

### 2.2.1 Data Partition

The given data set is divided into Development and Testing data set, with approximately 70:30 proportion using random variable. The distribution of Responder and Non Responder Class is verified in both the data sets, and ensured it's close to equal.

| | No. of Observations | No. of Non-Responders | No. of Responders | % of Responders | % of Non-Responders |
|---|---|---|---|---|---|
| **Dev Sample** | 13893 | 12141 | 1752 | 87.4 | 12.6 |
| **Testing Sample** | 6107 | 5347 | 760 | 87.5 | 12.4 |

### 2.2.2 Model Building

Initial CART model is built on the dev sample using the ideal control parameters and allowed the tree to grow. Later the tree is pruned by using the optimum CP value which is observed from the CP Plot and the model CP values. After the tree is pruned, the model is used to predict the class and score the predicted values and are added to the new columns in the data set. The observations and plots obtained are presented below.

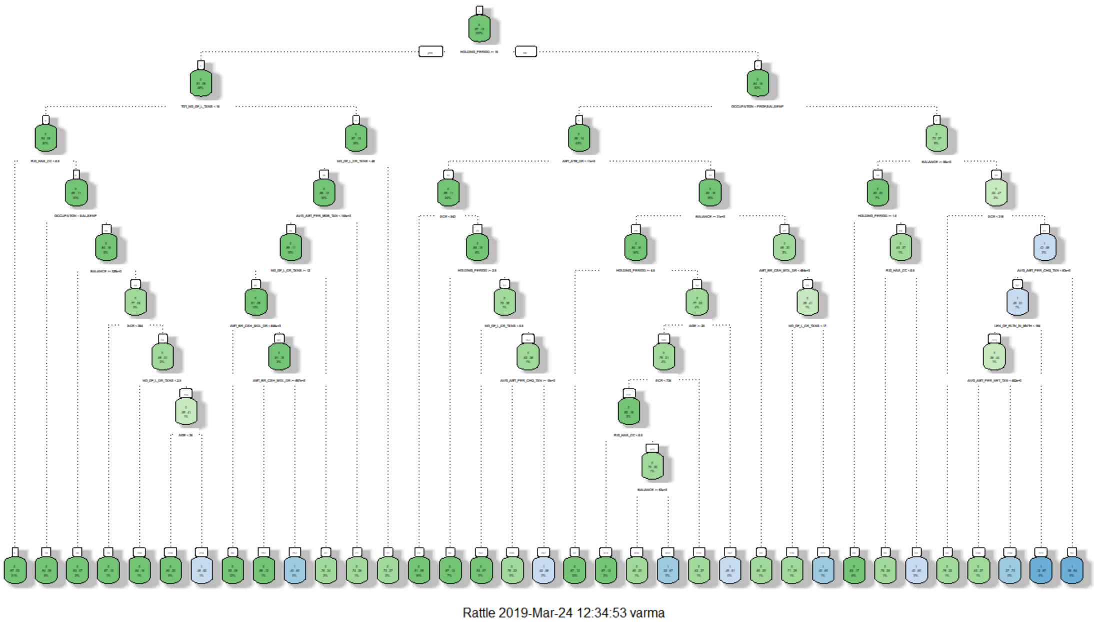
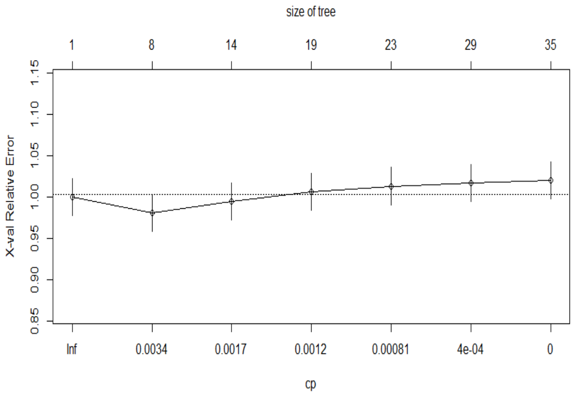
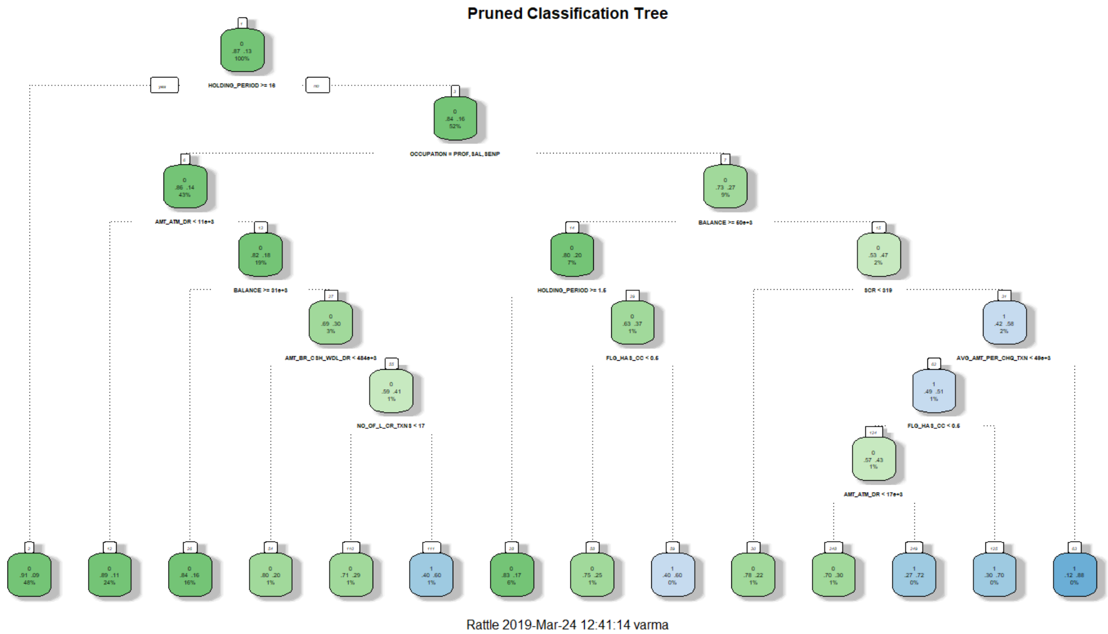



Figure.1 Initial Classification Tree




| CP | nsplit | rel error | xerror | xstd |
|---|---|---|---|---|
| 0.004994 | 0 | 1 | 1 | 0.022334 |
| **0.002283** | 7 | 0.96119 | 0.98116 | 0.022152 |
| 0.001256 | 13 | 0.94749 | 0.99543 | 0.02229 |
| 0.001142 | 18 | 0.94121 | 1.00685 | 0.022399 |
| 0.000571 | 22 | 0.93664 | 1.0137 | 0.022464 |
| 0.000285 | 28 | 0.93322 | 1.01712 | 0.022496 |
| 0 | 34 | 0.93151 | 1.02055 | 0.022529 |

CP Plot and the CP-values obtained for the Model




### 2.2.3 Model Performance

The following model performance measures are calculated on the development set to gauge the goodness of the model:

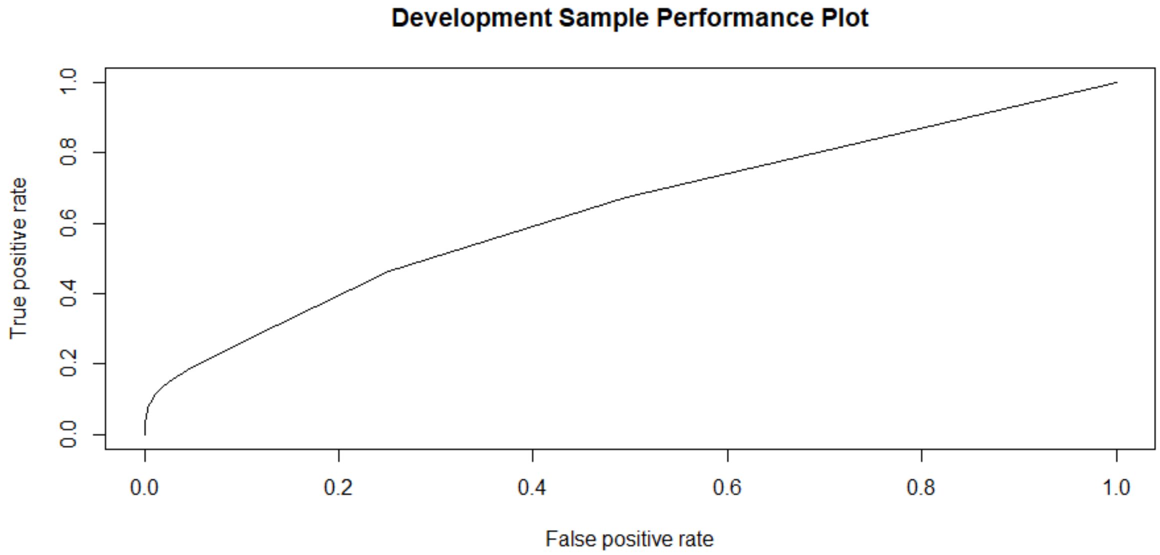
- Rank Ordering
- KS
- Area Under Curve (AUC)
- Gini Coefficient
- Classification Error

Based on the predicted score on the data set, it is assigned with the deciles. Rank table is obtained by executing the ranking code upon the data set. The response rate in top decile is around 28% and in top three deciles it is 67.52%. The KS is around 21.42%, indicating that the model is not a good model and might need to test it on more balanced data set.

The Rank table, Performance plot and the confusion matrix obtained from the model are presented below.

| deciles | cnt | cnt_resp | cnt_non_resp | rrate | cum_resp | cum_non_resp | cum_perct_resp | cum_perct_non_resp | ks |
|---|---|---|---|---|---|---|---|---|---|
| **10** | 1634 | 455 | 1179 | **27.85** | 455 | 1179 | 25.97 | 9.71 | 16.26 |
| **9** | 2238 | 361 | 1877 | 16.13 | 816 | 3056 | 46.58 | 25.17 | **21.41** |
| **8** | 3337 | 367 | 2970 | 11 | 1183 | 6026 | 67.52 | 49.63 | 17.89 |
| **5** | 6684 | 569 | 6115 | 8.51 | 1752 | 12141 | 100 | 100 | 0 |

Rank Table of the development sample




The Area under curve (AUC) of 63.5 % and Gini coefficient of 23.62 % are obtained from the model on the development sample and they also indicate that, model is not good. The Accuracy and Classification error rate are as below:

| | predict.class | |
|---|---|---|
| **TARGET** | 0 | 1 |
| **0** | 12060 | 81 |
| **1** | 1576 | 176 |

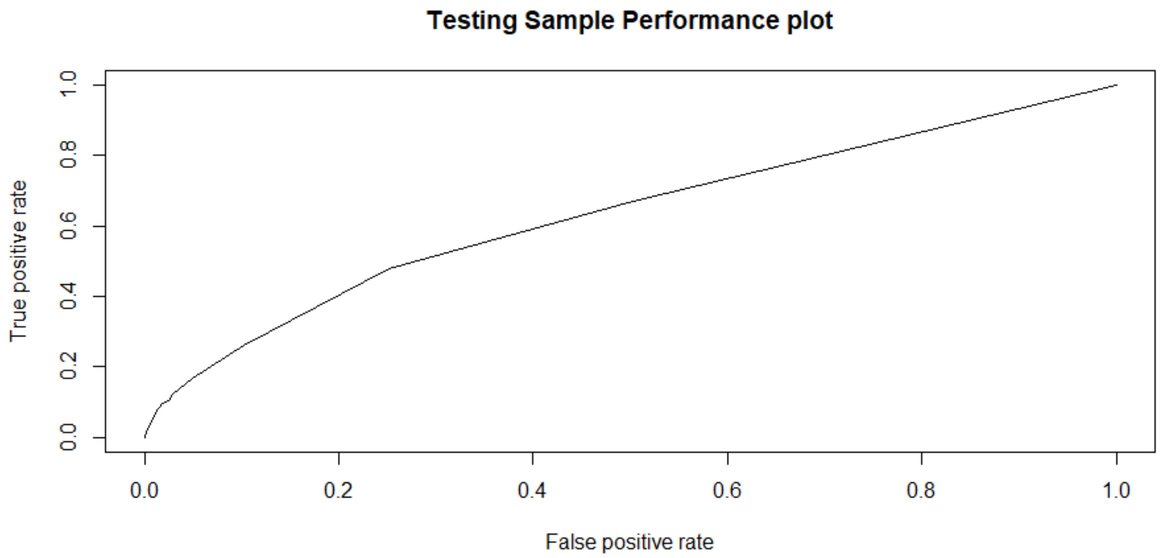
Accuracy = (12060+176)/13893 = 0.88073

Classification Error Rate = 1 – Accuracy = 0.11927

Now the model is used to predict on the testing sample and the observations are as follows:

| deciles | cnt | cnt_resp | cnt_non_resp | rrate | cum_resp | cum_non_resp | cum_perct_resp | cum_perct_non_resp | ks |
|---|---|---|---|---|---|---|---|---|---|
| **10** | 747 | 200 | 547 | **26.77** | 200 | 547 | 26.32 | 10.23 | 16.09 |
| **9** | 958 | 164 | 794 | 17.12 | 364 | 1341 | 47.89 | 25.08 | **22.81** |
| **8** | 1490 | 144 | 1346 | 9.66 | 508 | 2687 | 66.84 | 50.25 | 16.59 |
| **5** | 2912 | 252 | 2660 | 8.65 | 760 | 5347 | 100 | 100 | 0 |

The response rate in top decile is around 27% and in top three deciles it is 66.84%. The KS is around 22.81%, indicating that the model is not a good model.




The Area under curve (AUC) of 63.1 % and Gini coefficient of 24.65 % are obtained from the model on the testing sample and they also indicate that, model is not good. The Accuracy and Classification error rate are as below:

| | predict.class | |
|---|---|---|
| TARGET | 0 | 1 |
| 0 | 5284 | 63 |
| 1 | 706 | 54 |

Accuracy = (5284+54)/6107 = 0.874078

Classification Error Rate = 1 – Accuracy = 0.12592107

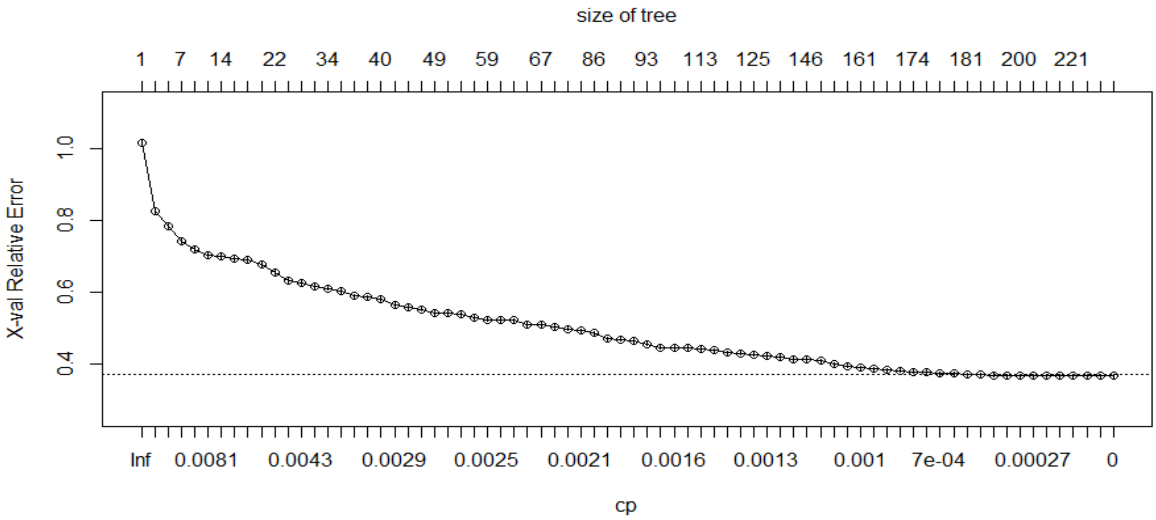
As the difference of the performance measure values of development sample and the testing sample are within the tolerance range of 5-10%, it is clear that model is not overfit model. Since the model performance measures indicates the classification model is not a good model, the development sample is balanced and model performance is measured on the new data set.

### 2.2.4 Model Building and Performance on Balanced Development Sample

The given data set is observed to be under sampled so to balance that the development data is over sampled using the ROSE algorithm, which adds synthetic data. Though Rose method is the best option to balance the data, over method is used as our target dependent variable is categorical.

The Model is built with the initial set of control parameters and then pruned with the CP value of 0.00030. After the tree is pruned, the model is used to predict the class and score the predicted values and are added to the new columns in the data set. The observations and plots obtained are presented below.

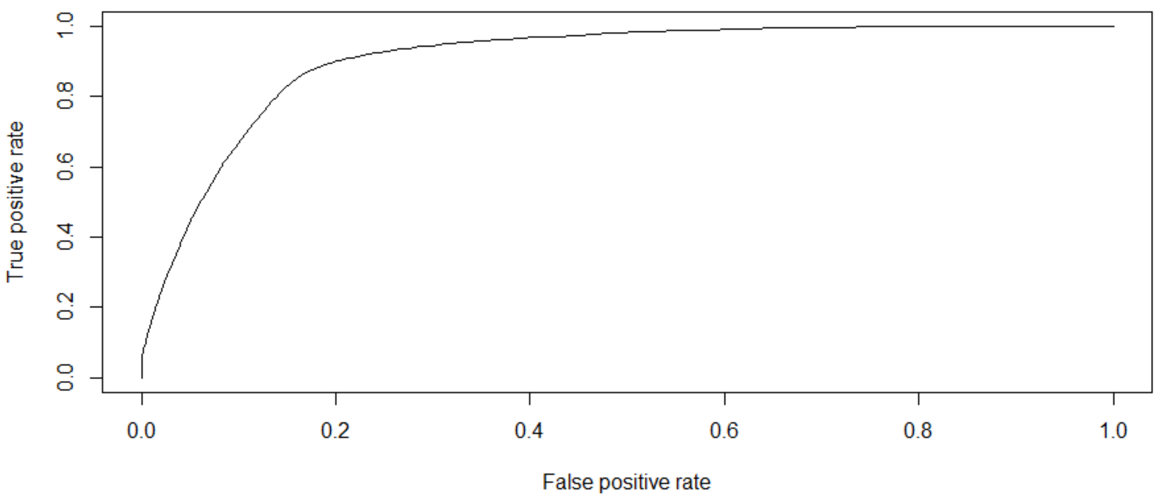
The response rate in top decile is around 93.38%. The KS is around 68.95%, indicating that the model is a very good model. The model is tested on the testing sample and the model performance measures are provided below.




CP plot of the Intial Model on Balanced Dev data set

| deciles | cnt | cnt_resp | cnt_non_resp | rrate | cum_resp | cum_non_resp | cum_perct_resp | cum_perct_non_resp | ks |
|---|---|---|---|---|---|---|---|---|---|
| **10** | 2536 | 2368 | 168 | **93.38** | 2368 | 168 | 19.5 | 1.38 | 18.12 |
| **9** | 2530 | 2220 | 310 | 87.75 | 4588 | 478 | 37.79 | 3.94 | 33.85 |
| **8** | 2533 | 2138 | 395 | 84.41 | 6726 | 873 | 55.4 | 7.19 | 48.21 |
| **7** | 2547 | 2025 | 522 | 79.51 | 8751 | 1395 | 72.08 | 11.49 | 60.59 |
| **6** | 2027 | 1521 | 506 | 75.04 | 10272 | 1901 | 84.61 | 15.66 | **68.95** |
| **5** | 2409 | 1074 | 1335 | 44.58 | 11346 | 3236 | 93.45 | 26.65 | 66.8 |
| **4** | 2703 | 486 | 2217 | 17.98 | 11832 | 5453 | 97.45 | 44.91 | 52.54 |
| **3** | 2169 | 216 | 1953 | 9.96 | 12048 | 7406 | 99.23 | 61 | 38.23 |
| **2** | 2486 | 93 | 2393 | 3.74 | 12141 | 9799 | 100 | 80.71 | 19.29 |
| **1** | 2342 | 0 | 2342 | 0 | 12141 | 12141 | 100 | 100 | 0 |

Rank Table of the Pruned Model on Balanced Dev Data set




Performance Plot of Pruned Model on Balanced Dev Data set

The Area under curve (AUC) of 90.7 % and Gini coefficient of 40.70 % are obtained from the model on the balanced development sample and they also indicate that, model is very good. The Accuracy and Classification error rate are as below:

| | predict.class | |
|---|---|---|
| **TARGET** | 0 | 1 |
| **0** | 9928 | 2213 |
| **1** | 1405 | 10736 |

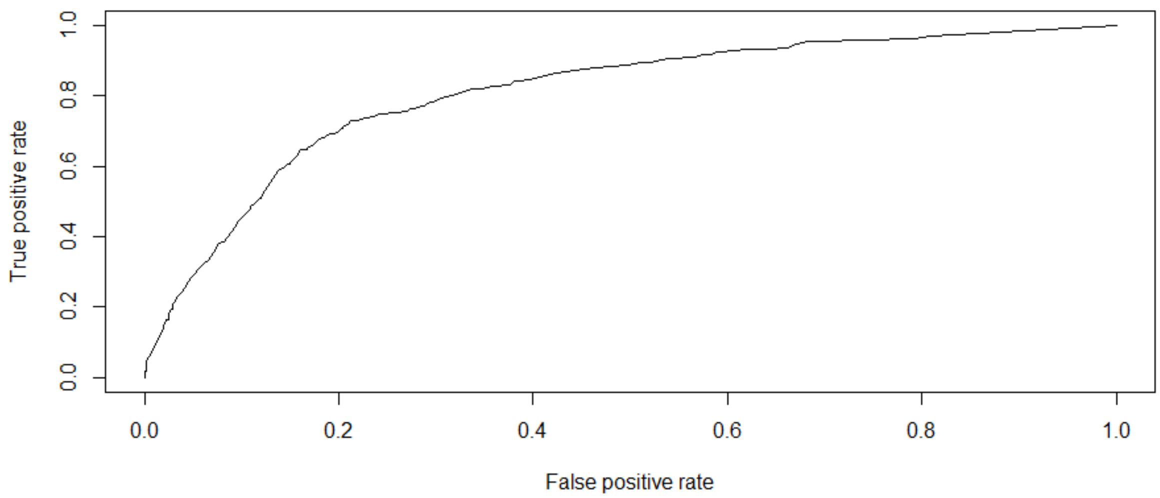
Accuracy = (9928+10736)/24282 = 0.85100

Classification Error Rate = 1 – Accuracy = 0.1489

The response rate in top decile from the rank table of the model on testing data set is around 41.59%. The KS is around 50.65%, indicating that the model is a good model.

| deciles | cnt | cnt_resp | cnt_non_resp | rrate | cum_resp | cum_non_resp | cum_perct_resp | cum_perct_non_resp | ks |
|---|---|---|---|---|---|---|---|---|---|
| **10** | 618 | 257 | 361 | **41.59** | 257 | 361 | 33.82 | 6.75 | 27.07 |
| **9** | 618 | 202 | 416 | 32.69 | 459 | 777 | 60.39 | 14.53 | 45.86 |
| **8** | 628 | 110 | 518 | 17.52 | 569 | 1295 | 74.87 | 24.22 | **50.65** |
| **7** | 616 | 54 | 562 | 8.77 | 623 | 1857 | 81.97 | 34.73 | 47.24 |
| **6** | 674 | 47 | 627 | 6.97 | 670 | 2484 | 88.16 | 46.46 | 41.7 |
| **5** | 565 | 22 | 543 | 3.89 | 692 | 3027 | 91.05 | 56.61 | 34.44 |
| **4** | 564 | 24 | 540 | 4.26 | 716 | 3567 | 94.21 | 66.71 | 27.5 |
| **3** | 679 | 15 | 664 | 2.21 | 731 | 4231 | 96.18 | 79.13 | 17.05 |
| **2** | 1145 | 29 | 1116 | 2.53 | 760 | 5347 | 100 | 100 | 0 |

Rank Table of the Pruned Model on Testing Data set




Performance Plot of Pruned Model on Testing Data set

The Area under curve (AUC) of 80.84 % and Gini coefficient of 58.10 % are obtained from the model on the testing sample and they also indicate that, model is very good. The Accuracy and Classification error rate are as below:

| | predict.class | |
|---|---|---|
| **TARGET** | 0 | 1 |
| **0** | 4201 | 1146 |
| **1** | 205 | 555 |

Accuracy = (4201+555)/6107 = 0.7787

Classification Error Rate = 1 – Accuracy = 0.2213

The Accuracy of the model on the balanced development sample is more than that of the testing sample and the classification error rate of the testing sample is more indicatin g the model built on the balanced data set is not performed as expected in the unseen data set.

### 2.2.5 Conclusion

The CART Model built on the unbalanced data set has come out to be not good model(KS-22) and also not a overfit model as the difference of performance measures on dev and testing sample are under the tolerance limit. Whereas the model built on the balanced data set outperformed and resulted in almost perfect model (KS-68), but the difference in some performance measures are in the range of 10-20% indicating slight deviation. It might be because of the testing sample being unbalanced and under sized compared to the balanced development sample. Overall it has performed well on the testing sample with KS of 50.65 and with Gini coefficient of around 58%.

## 2.3 Random Forest Model

Random Forests is a versatile machine learning method capable of performing both regression and classification tasks. It also undertakes dimensional reduction methods, tre ats missing values, outlier values and other essential steps of data exploration, and does a fairly good job.

Random forests improve predictive accuracy by generating a large number of bootstrapped trees (based on random samples of variables), classifying a case using each tree in this new "forest", and deciding a final predicted outcome by combining the results across all of the trees (an average in regression, a majority vote in classification).

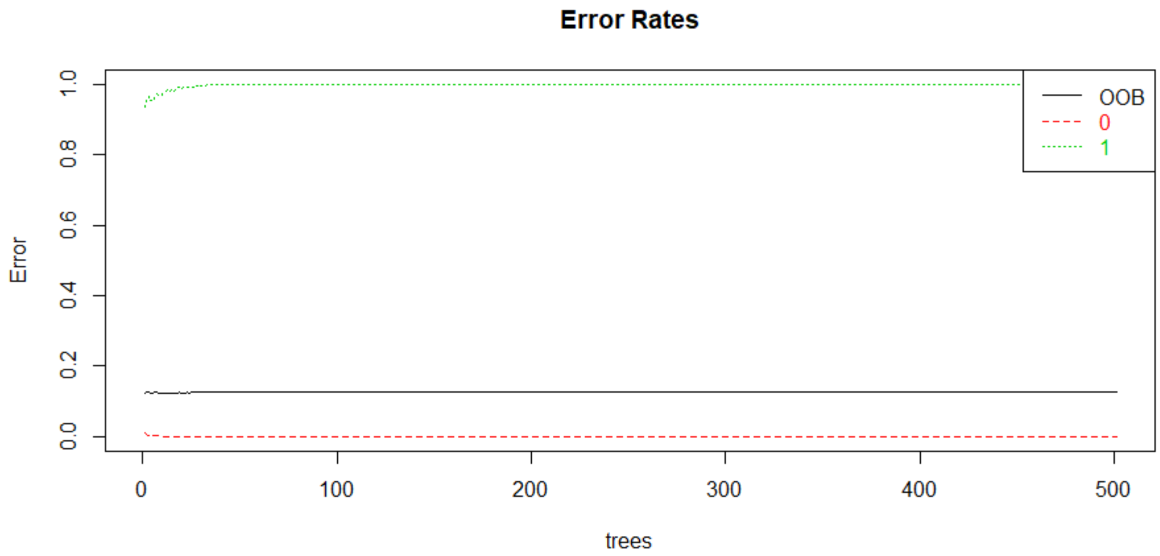
### 2.3.1 Data Partition

The given data set is divided into Development and Testing data set, with approximately 70:30 proportion using random variable. The distribution of Responder and Non Responder Class is verified in both the data sets, and ensured it’s close to equal.

### 2.3.2 Model Building

Initial Random Forest model is built on the dev sample using the ideal parameters for mtry, nodesize and ntree. Based on the error rate optimum value for ntree is found out and the model is tuned to get the optimum mtry value. The Final tuned model is built using the optimum values and it is used to predict the class and score the predicted values.

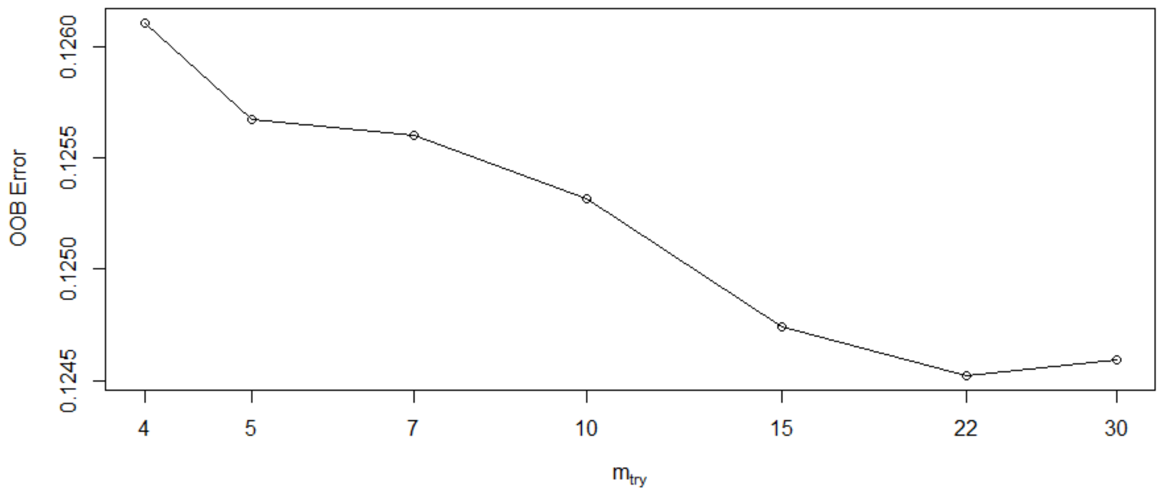
The observations and plots of the model are presented below:




```
randomForest(formula = as.factor(TARGET) ~ ., data = PL_RF_data.dev,     ntree = 501,
mtry = 5, nodesize = 250, importance = TRUE)
          Type of random forest: classification
              Number of trees: 501
No. of variables tried at each split: 5

     OOB estimate of  error rate: 12.6%
Confusion matrix:
   0 1 class.error
0 12141 0   0.0000000
1  1751 1   0.9994292
```

The output of the Initial model gives OOB error rate of 12.6%. From the error rate values the optimum value of trees is chosen as 121 and the model is tuned with step factor value of 1.5. The result of the tuned Random Forest model is captured in the below graph.



The optimum value of mtry obtained from the result is chosen as 22 and the model is built again with these values. The result of the final model is as below:

```
randomForest(formula = as.factor(TARGET) ~ ., data = PL_RF_data.dev,      ntree = 121, mtr
y = 22, nodesize = 250, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 121
No. of variables tried at each split: 22

        OOB estimate of  error rate: 12.43%
Confusion matrix:
      0  1  class.error
0 12138  3 0.0002470966
1  1724 28 0.9840182648
```

### 2.3.3 Model Performance

The following model performance measures are calculated on the development set to gauge the goodness of the model:

- Rank Ordering
- KS
- Area Under Curve (AUC)
- Gini Coefficient
- Classification Error

Based on the predicted score on the data set, it is assigned with the deciles. Rank table is obtained by executing the ranking code upon the data set. The response rate in top decile is around 58.74% and cumulative response rate in top three deciles is 89%. The KS is around 67%, indicating that the model is a very good model.

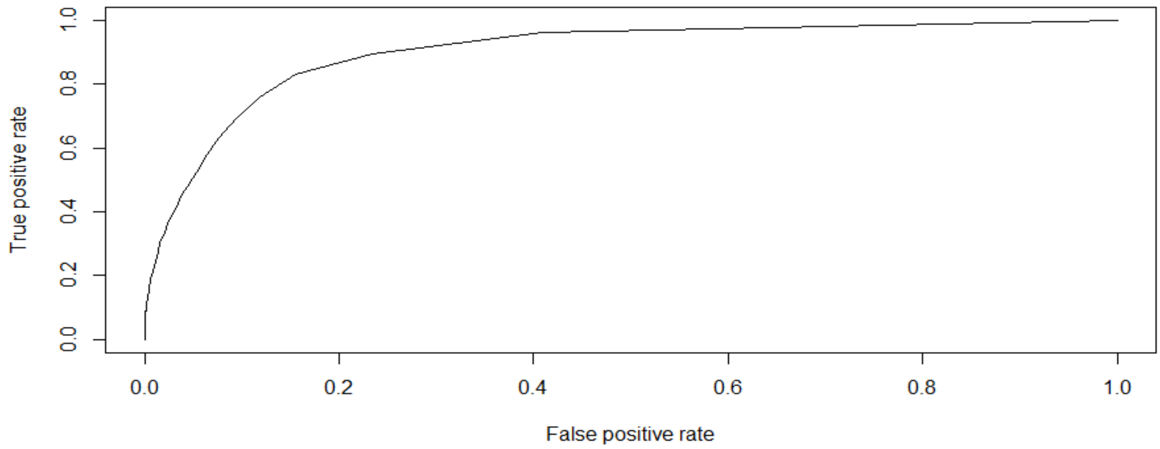
| deciles | cnt | cnt_resp | cnt_non_resp | rrate | cum_resp | cum_non_resp | cum_perct_resp | cum_perct_non_resp | ks |
|---|---|---|---|---|---|---|---|---|---|
| **10** | 1561 | 917 | 644 | **58.74** | 917 | 644 | 0.52 | 0.05 | 0.47 |
| **9** | 1779 | 539 | 1240 | 30.3 | 1456 | 1884 | 0.83 | 0.16 | **0.67** |
| **8** | 1070 | 112 | 958 | 10.47 | 1568 | 2842 | 0.89 | 0.23 | 0.66 |
| **7** | 2246 | 120 | 2126 | 5.34 | 1688 | 4968 | 0.96 | 0.41 | 0.55 |
| **6** | 7237 | 64 | 7173 | 0.88 | 1752 | 12141 | 1 | 1 | 0 |

The Area under curve (AUC) of 90.2% and Gini coefficient of 81% are obtained from the model on the development sample and they also indicate that, model is very good. The Accuracy and Classification error rate are as below:

| | predict.class | |
|---|---|---|
| **TARGET** | 0 | 1 |
| **0** | 12141 | 0 |
| **1** | 1713 | 39 |

Accuracy = (12141+39)/13893 = 0.8767

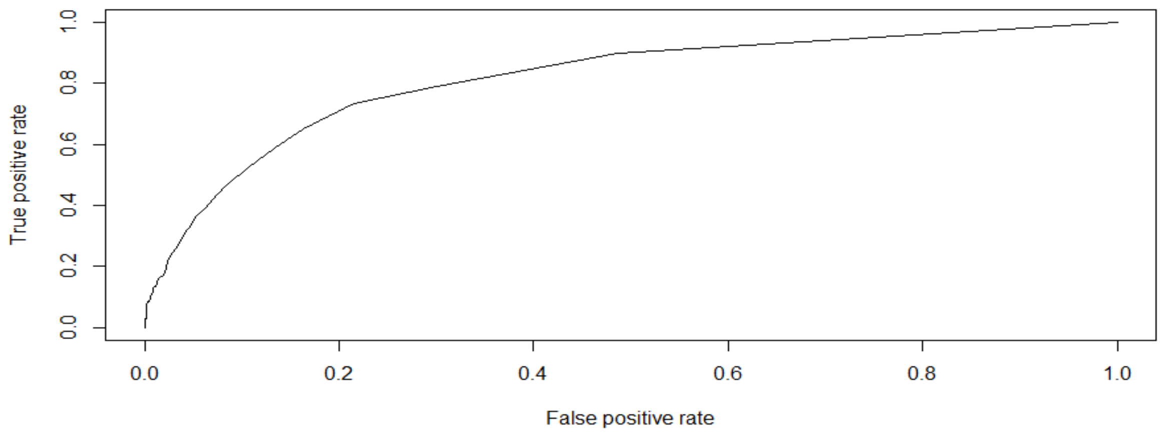
Classification Error Rate = 1 – Accuracy = 0.1233




Now the model is used to predict on the testing sample and the observations are as follows:

| deciles | cnt | cnt_resp | cnt_non_resp | rrate | cum_resp | cum_non_resp | cum_perct_resp | cum_perct_non_resp | ks |
|---|---|---|---|---|---|---|---|---|---|
| **10** | 632 | 299 | 333 | **47.31** | 299 | 333 | 0.39 | 0.06 | 0.33 |
| **9** | 735 | 197 | 538 | 26.8 | 496 | 871 | 0.65 | 0.16 | **0.49** |
| **8** | 843 | 105 | 738 | 12.46 | 601 | 1609 | 0.79 | 0.3 | 0.49 |
| **7** | 1070 | 82 | 988 | 7.66 | 683 | 2597 | 0.9 | 0.49 | 0.41 |
| **5** | 2827 | 77 | 2750 | 2.72 | 760 | 5347 | 1 | 1 | 0 |

The response rate in top decile is around 47.31% and the cumulative response in top three deciles is 79%. The KS is around 50%, indicating that the model is a good model. The performance plot on the testing sample is as below:



The Area under curve (AUC) of 81.49 % and Gini coefficient of 78.3 % are obtained from the model on the testing sample and they also indicate that, model is good. The Accuracy and Classification error rate are as below:

| | predict.class | |
|---|---|---|
| **TARGET** | 0 | 1 |
| **0** | 5347 | 0 |
| **1** | 752 | 8 |

Accuracy = (5347+8)/6107 = 0.8768

Classification Error Rate = 1 – Accuracy = 0.1231

As the difference of the performance measure values of development sample and the testing sample are within the tolerance range of 5-10% except for ks which is around 16%, it can be inferred that model is not much deviating and not much overfit.

### 2.3.4 Conclusion

The Random Forest Model built on the data set has come out to be a very good model(KS-67) and also not a overfit model as the difference of performance measures on dev and testing sample are under the tolerance limit. Though the data set is unbalanced Random Forest has performed better than the Classification Tree model.

## 2.4 Neural Network Model

Neural networks are a set of algorithms, modelled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated.

Neural networks help us cluster and classify. We can think of them as a clustering and classification layer on top of the data we store and manage. They help to group unlabeled data according to similarities among the example inputs, and they classify data when they have a labeled dataset to train on. (Neural networks can also extract features that are fed to other algorithms for clustering and classification; so we can think of deep neural networks as components of larger machine-learning applications involving algorithms for reinforcement learning, classification and regression.)

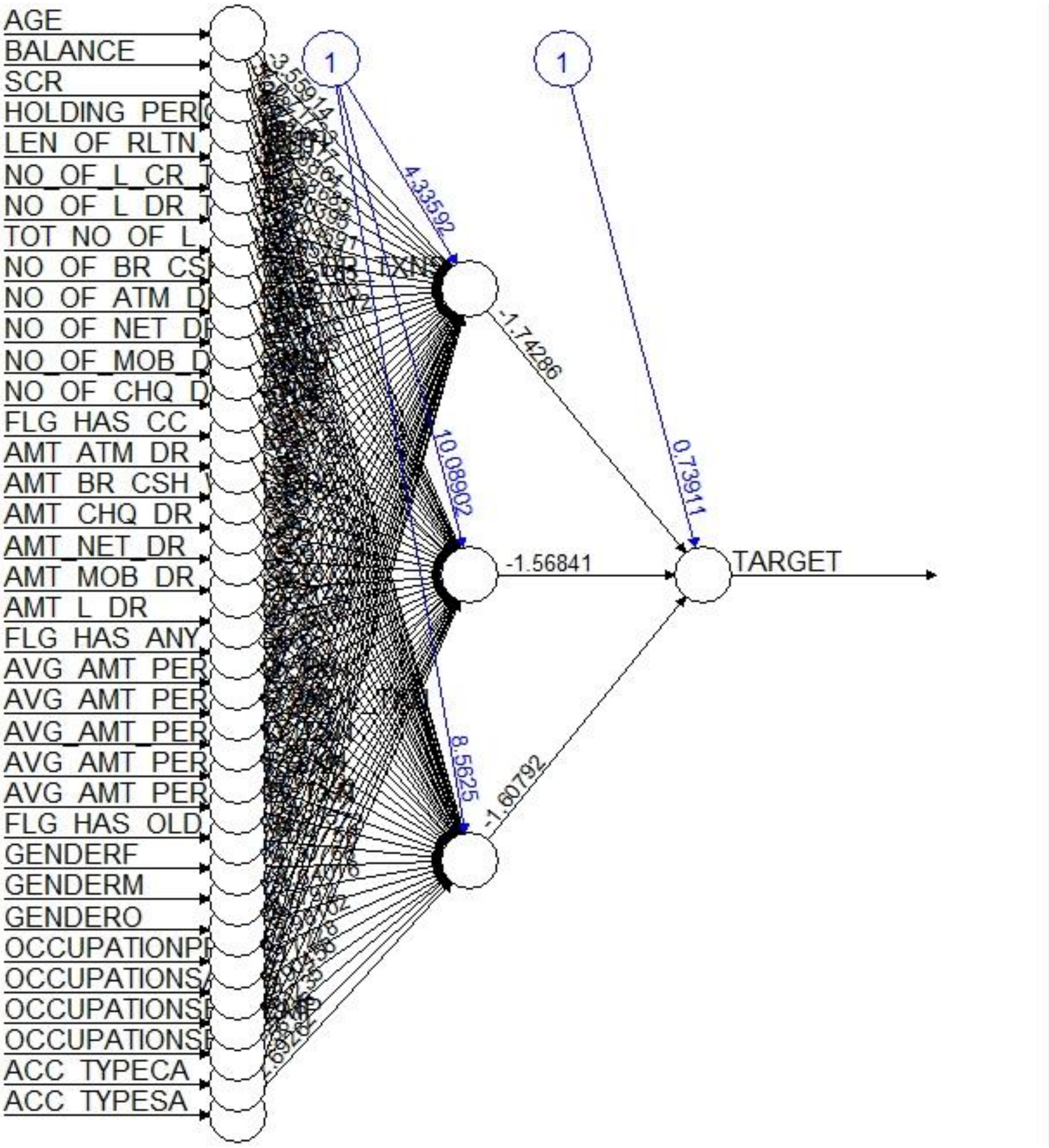
### 2.4.1 Data Partition

The given data set is divided into Development and Testing data set, with approximately 70:30 proportion using random variable. The distribution of Responder and Non Responder Class is verified in both the data sets, and ensured it’s close to equal.

### 2.4.2 Model Building

Before the Neural Network model is built on the dev sample, all the categorical variables are converted into dummy variables and the obtained data set is scaled as it can be performed only on numeric variables. Now the model is built with all the independent variables and by setting the appropriate values for all the parameters required for the neural net method.

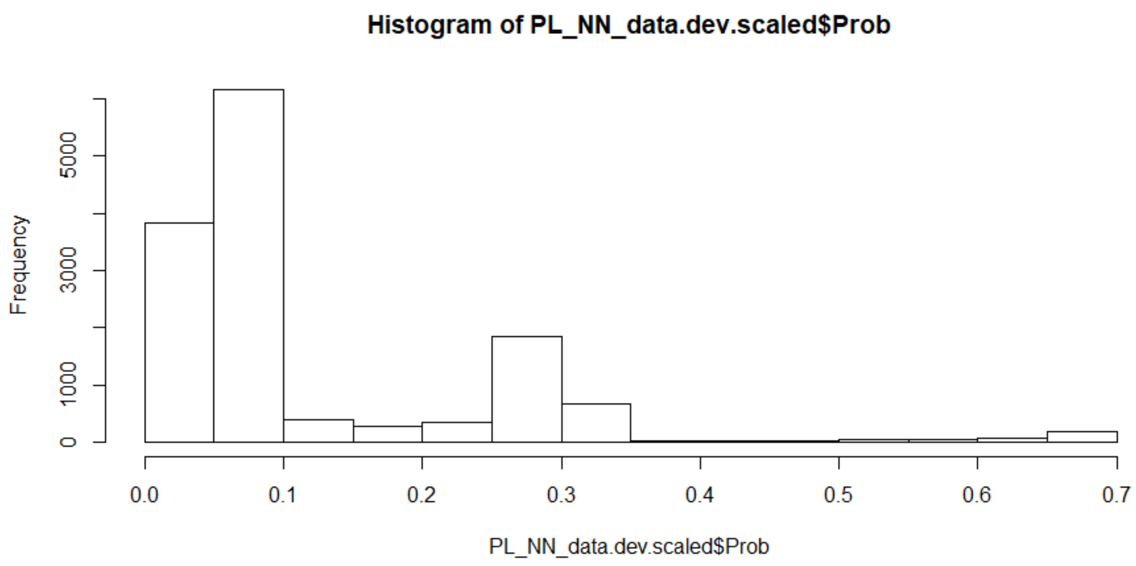
The method is executed multiple times to confirm the convergence with the initial set of values for the parameters. The converged model is used to predict the class and score the predicted values. The observations and plots of the model are as below:




The probability values for the result from the neural net model is against value of '1' are assigned to the data set and the distribution of that estimated probability is as below. The distribution looks fine and the probability of 0.6768 in the 100% range.

```
0%         1%         5%          10%        25%        50%
0.01506671 0.01506671 0.01506709 0.01509074 0.03286275 0.07643855

75%        90%        95%        98%         99%        100%
0.15539278 0.29455431 0.30377383 0.54317019  0.66370753 0.67680188
```



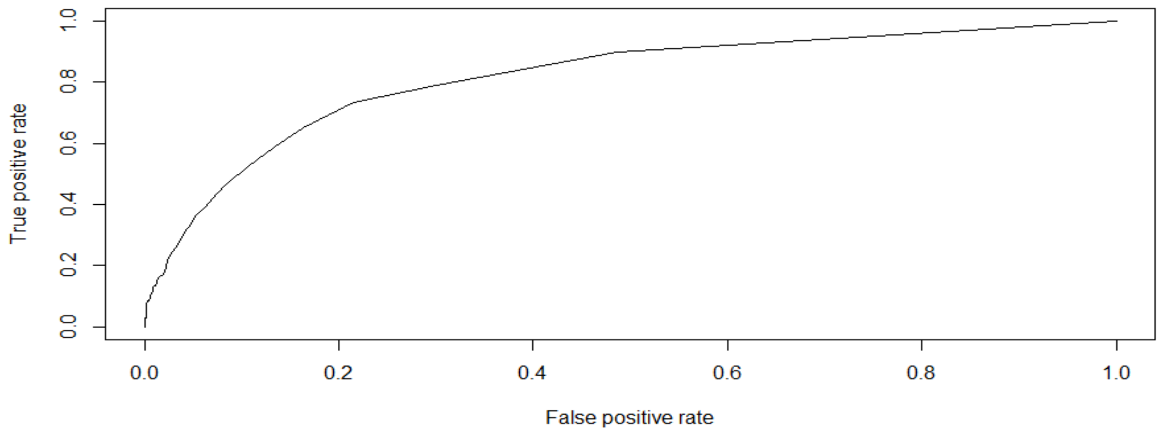

### 2.4.3 Model Performance

The following model performance measures are calculated on the development set to gauge the goodness of the model:

- Rank Ordering
- KS
- Area Under Curve (AUC)
- Gini Coefficient
- Classification Error

Based on the predicted score on the data set, it is assigned with the deciles. Rank table is obtained by executing the ranking code upon the data set. The response rate in top decile is around 40% and cumulative response rate in top three deciles is 63%. The KS is around 38%, from the table indicating that the model is a good model and scope for improvement.

| deciles | cnt | cnt_resp | cnt_non_resp | rrate | cum_resp | cum_non_resp | cum_perct_resp | cum_perct_non_resp | ks |
|---|---|---|---|---|---|---|---|---|---|
| **10** | 1392 | 552 | 840 | **40.00%** | 552 | 840 | 32.00% | 7.00% | 0.25 |
| **9** | 1388 | 270 | 1118 | 19.00% | 822 | 1958 | 47.00% | 16.00% | 0.31 |
| **8** | 1388 | 277 | 1111 | 20.00% | 1099 | 3069 | 63.00% | 25.00% | **0.38** |
| **7** | 1389 | 153 | 1236 | 11.00% | 1252 | 4305 | 71.00% | 35.00% | 0.36 |
| **6** | 1390 | 89 | 1301 | 6.00% | 1341 | 5606 | 77.00% | 46.00% | 0.31 |
| **5** | 1391 | 109 | 1282 | 8.00% | 1450 | 6888 | 83.00% | 57.00% | 0.26 |
| **4** | 1388 | 98 | 1290 | 7.00% | 1548 | 8178 | 88.00% | 67.00% | 0.21 |
| **3** | 1388 | 104 | 1284 | 7.00% | 1652 | 9462 | 94.00% | 78.00% | 0.16 |
| **2** | 1390 | 55 | 1335 | 4.00% | 1707 | 10797 | 97.00% | 89.00% | 0.08 |
| **1** | 1389 | 45 | 1344 | 3.00% | 1752 | 12141 | 100.00% | 100.00% | 0 |




The Area under curve (AUC) of 81.49% and Gini coefficient of 49.63% are obtained from the model on the development sample and they also indicate that, model is very good. The Accuracy and Classification error rate are as below:

| | predict.class | |
|---|---|---|
| **TARGET** | 0 | 1 |
| **0** | 11999 | 142 |
| **1** | 1463 | 289 |

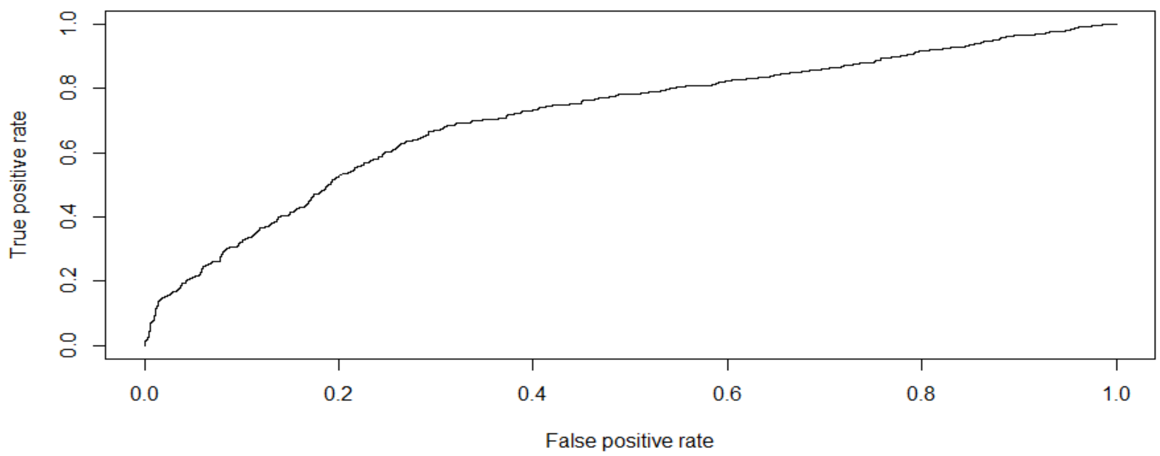
Accuracy = (11999+289)/13893 = 0.8847

Classification Error Rate = 1 – Accuracy = 0.1153

Now the model is used to predict on the testing sample and the observations are as follows:

| deciles | cnt | cnt_resp | cnt_non_resp | rrate | cum_resp | cum_non_resp | cum_perct_resp | cum_perct_non_resp | ks |
|---|---|---|---|---|---|---|---|---|---|
| **10** | 611 | 200 | 411 | **33.00%** | 200 | 411 | 26.00% | 8.00% | 0.18 |
| **9** | 611 | 133 | 478 | 22.00% | 333 | 889 | 44.00% | 17.00% | 0.27 |
| **8** | 610 | 130 | 480 | 21.00% | 463 | 1369 | 61.00% | 26.00% | **0.35** |
| **7** | 611 | 73 | 538 | 12.00% | 536 | 1907 | 71.00% | 36.00% | 0.35 |
| **6** | 611 | 45 | 566 | 7.00% | 581 | 2473 | 76.00% | 46.00% | 0.3 |
| **5** | 610 | 34 | 576 | 6.00% | 615 | 3049 | 81.00% | 57.00% | 0.24 |
| **4** | 611 | 33 | 578 | 5.00% | 648 | 3627 | 85.00% | 68.00% | 0.17 |
| **3** | 610 | 39 | 571 | 6.00% | 687 | 4198 | 90.00% | 79.00% | 0.11 |
| **2** | 612 | 44 | 568 | 7.00% | 731 | 4766 | 96.00% | 89.00% | 0.07 |
| **1** | 610 | 29 | 581 | 5.00% | 760 | 5347 | 100.00% | 100.00% | 0 |

The response rate in top decile is around 33% and the cumulative response in top three deciles is 61%. The KS is around 35%, indicating that the model is a good model. The performance plot on the testing sample is as below:




The Area under curve (AUC) of 71.36 % and Gini coefficient of 49.21 % are obtained from the model on the testing sample and they also indicate that, model is good. The Accuracy and Classification error rate are as below:

| | predict.class | |
|---|---|---|
| **TARGET** | 0 | 1 |
| **0** | 5271 | 76 |
| **1** | 656 | 104 |

Accuracy = (5271+104)/6107 = 0.8801

Classification Error Rate = 1 – Accuracy = 0.1199

As the difference of the performance measure values of development sample and the testing sample are within the tolerance range of 5-10% the model is not overfit and still there is chance of improvement.

### 2.4.4 Conclusion

The Neural Net Model built on the data set has come out to be a good model(KS-38) and also not a overfit model as the difference of performance measures on dev and testing sample are under the tolerance limit. Though the data set is unbalanced Neural Network Model has performed better than the Classification Tree model and still could be improved by balancing the data set.

## 3. Conclusion – Model Comparison

The main objective of the project was to develop a predictive model to predict if MyBank customers will respond positively to a promotion or an offer using tools of Machine Learning. In this context, the key parameter for model evaluation was 'Accuracy', i.e., the proportion of the total number of predictions that were correct (i.e. % of the customers that were correctly predicted).

The predictive models were developed using the following Machine Learning techniques: Classification Tree – CART, Random Forest and Neural Network. The overview of the performance of all the models on accuracy, over-fitting and other model performance measures is provided below:

### Classification Tree Model

| Measures | Train | Test | %Deviation |
|---|---|---|---|
| KS | 68.95 | 50.65% | 18% |
| AUC | 90.7% | 80.84% | 9.86% |
| Gini | 40.7% | 58.1% | 17.4% |
| Accuracy | 85.1% | 77.87% | 7.23% |
| CeR | 14.91% | 22.13% | 7.22% |

### Radom Forest Model

| Measures | Train | Test | %Deviation |
|---|---|---|---|
| KS | 67% | 49% | 18% |
| AUC | 90.2% | 81.49% | 8.71% |
| Gini | 81% | 78.3% | 2.7% |
| Accuracy | 87.67 | 87.68% | 0.01% |
| CeR | 12.33% | 12.31% | 0.02% |

### Artificial Neural Network

| Measures | Train | Test | %Deviation |
|---|---|---|---|
| KS | 38% | 35% | 3% |
| AUC | 81.49% | 71.36% | 10.13% |
| Gini | 49.63% | 49.21% | 0.42% |
| Accuracy | 88.47% | 88.01% | 0.46% |
| CeR | 11.53% | 11.99% | 0.46% |

**Interpretation:**

- The CART method has given poor performance compared to Random Forest and Neural Network on the unbalanced data set. Looking at the percentage deviation between Training and Testing Dataset, it looks like the Model is over fit.
- The Random Forest method has the best performance (best accuracy) among all the three models. The percentage deviation between Training and Testing Dataset also is reasonably under control, suggesting a robust model.
- Neural Network has given relatively secondary performance compared to Random Forest, however, better than CART. However, the percentage deviation between Training and Testing Data set is minimal among three models.

To conclude, Random Forest Model seems to be the best model understandably as it handles the outliers, deviations and multiple data type variables very well and has given the better accuracy % and reasonable deviations.

# 4. Appendix A – Source Code

```
#set up of working directory
setwd("D:/BACP Program/R Directory")
getwd()

library(moments)
library(forecast)
library(car)

#preliminary Analysis

##Importing the data
PL_data <- read.csv("PL_XSELL.csv",sep=",",header=T)

#dimension of the data set
dim(PL_data)

##Viewing the structure of the data (data types- class)
str(PL_data)

#View(PL_data)
attach(PL_data)

##Summary of the data
nrow(PL_data)
#There are 20000 rows in all, with the following characteristics:

#check top 10 and bottom 10 observations
#head(PL_data)
#tail(PL_data)

#checking for missing values
```

```
colSums(is.na(PL_data))
#No missing values

#Checking individual columns - Exploratory Data Analysis
summary(factor(TARGET))
#2512 customers have responded for personal loan.

prop.table(table(TARGET))
#12.56 % customers have responded.
#87.44 % customers have not responded.

summary(AGE)
#The minimum age of customer is 21, while the maximum is 55.
#Between these two values, the age looks to be uniformly spread.
hist(AGE)
boxplot(AGE)
#Normal

summary(GENDER)
#There are more male customers than females and others.
table(GENDER, TARGET)
#More Male customers have responded.
chisq.test(TARGET, GENDER)
#Gender significant

summary(BALANCE)
hist(BALANCE)
boxplot(BALANCE)
BoxCox.lambda(BALANCE)
#No Transformation As it contains '0' value

summary(OCCUPATION)
# saliried customers are more in number
table(TARGET, OCCUPATION)
#More Self-employed customers have responded.
chisq.test(TARGET, OCCUPATION)
#significant

summary(AGE_BKT)
#More number of customers who responded are in the age bracket of 41-45 and
in the range of 26-45
table(TARGET, AGE_BKT)
#since age is already given, neglecting AGE_BKT-[7]

summary(SCR)
#score varies from min 100 to max 999
boxplot(SCR)
hist(SCR)
BoxCox.lambda(SCR)
#Transformation
```

```
PL_data$SCR <- 1/sqrt(SCR)

summary(HOLDING_PERIOD)
hist(HOLDING_PERIOD)
boxplot(HOLDING_PERIOD)
#Normal

summary(ACC_TYPE)
#customers having savings account are more so are the responded one's
table(TARGET, ACC_TYPE)
chisq.test(TARGET, ACC_TYPE)
#significant

summary(ACC_OP_DATE)
#neglecting ACC_OP_DATE-[11]

summary(LEN_OF_RLTN_IN_MNTH)
hist(LEN_OF_RLTN_IN_MNTH)
boxplot(LEN_OF_RLTN_IN_MNTH)
#Normal

summary(NO_OF_L_CR_TXNS)
hist(NO_OF_L_CR_TXNS)
boxplot(NO_OF_L_CR_TXNS)
#Right-skewed, more observation fall in the range of 0-10
BoxCox.lambda(NO_OF_L_CR_TXNS)
#Even though the boxCox method indicates to perform log transformation,
#since it would add infinity to the feature, transformation is avoided

summary(NO_OF_L_DR_TXNS)
hist(NO_OF_L_DR_TXNS)
boxplot(NO_OF_L_DR_TXNS)
#Right-skewed, most on the lower side and from 0-5

summary(TOT_NO_OF_L_TXNS)
hist(TOT_NO_OF_L_TXNS)
boxplot(TOT_NO_OF_L_TXNS)
#Right-skewed, More observations  in the range of 0-10

summary(NO_OF_BR_CSH_WDL_DR_TXNS)
#Branch cashwithdrawals varies from 0 to 15
hist(NO_OF_BR_CSH_WDL_DR_TXNS)
boxplot(NO_OF_BR_CSH_WDL_DR_TXNS)
#Right-skewed, Maximum withdrawals are from 0-5 times

summary(NO_OF_ATM_DR_TXNS)
hist(NO_OF_ATM_DR_TXNS)
boxplot(NO_OF_ATM_DR_TXNS)
#Maximun customers had done 0-4 ATM Transactions
```

```
summary(NO_OF_NET_DR_TXNS)
hist(NO_OF_NET_DR_TXNS)
boxplot(NO_OF_NET_DR_TXNS)
#Most of the customers had again 0-4 Net debit Transactions

summary(NO_OF_MOB_DR_TXNS)
hist(NO_OF_MOB_DR_TXNS)
boxplot(NO_OF_MOB_DR_TXNS)
#Mostly 0-2 Mobile Transactions

summary(NO_OF_CHQ_DR_TXNS)
hist(NO_OF_CHQ_DR_TXNS)
boxplot(NO_OF_CHQ_DR_TXNS)
#Right-Skewed, Mostly falling on the lower side

summary(FLG_HAS_CC)
hist(FLG_HAS_CC)
boxplot(FLG_HAS_CC)
#More customers doesn't have creditcard
chisq.test(TARGET, FLG_HAS_CC)
#significant

summary(AMT_ATM_DR)
hist(AMT_ATM_DR)
boxplot(AMT_ATM_DR)
#Right-skewed, Most amount drawn is from 0-50000

summary(AMT_BR_CSH_WDL_DR)
hist(AMT_BR_CSH_WDL_DR)
boxplot(AMT_BR_CSH_WDL_DR)
#evenly distributed across most of the data

summary(AMT_CHQ_DR)
hist(AMT_CHQ_DR)
boxplot(AMT_CHQ_DR)
#Most of the observations on th elower side

summary(AMT_NET_DR)
hist(AMT_NET_DR)
boxplot(AMT_NET_DR)
#Most obersvations on the lower side and the rest are evenly distributed

summary(AMT_MOB_DR)
hist(AMT_MOB_DR)
boxplot(AMT_MOB_DR)
#Most obersvations on the lower side and the rest are evenly distributed

summary(AMT_L_DR)
hist(AMT_L_DR)
boxplot(AMT_L_DR)
```

```
#Most of the amount debited falls in 0-2 lakhs

summary(FLG_HAS_ANY_CHGS)
hist(FLG_HAS_ANY_CHGS)
boxplot(FLG_HAS_ANY_CHGS)
#More customers doesn't have any bank charges
chisq.test(TARGET, FLG_HAS_ANY_CHGS)
#significant

summary(AMT_OTH_BK_ATM_USG_CHGS)
hist(AMT_OTH_BK_ATM_USG_CHGS)
boxplot(AMT_OTH_BK_ATM_USG_CHGS)
#Mostly on the lower side as atm transactions are also in the limit of 4
#Almost all observations have value 0, neglecting - [29]

summary(AMT_MIN_BAL_NMC_CHGS)
hist(AMT_MIN_BAL_NMC_CHGS)
boxplot(AMT_MIN_BAL_NMC_CHGS)
#Almost all obesrvations have value 0, neglecting - [30]

summary(NO_OF_IW_CHQ_BNC_TXNS)
hist(NO_OF_IW_CHQ_BNC_TXNS)
boxplot(NO_OF_IW_CHQ_BNC_TXNS)
#Almost all observations have value 0, neglecting - [31]

summary(NO_OF_OW_CHQ_BNC_TXNS)
hist(NO_OF_OW_CHQ_BNC_TXNS)
boxplot(NO_OF_OW_CHQ_BNC_TXNS)
#Almost all observations have value 0, neglecting - [32]

summary(AVG_AMT_PER_ATM_TXN)
hist(AVG_AMT_PER_ATM_TXN)
boxplot(AVG_AMT_PER_ATM_TXN)
#Falls mostly on the lower side, distributed evenly afterwards

summary(AVG_AMT_PER_CSH_WDL_TXN)
hist(AVG_AMT_PER_CSH_WDL_TXN)
boxplot(AVG_AMT_PER_CSH_WDL_TXN)
#skewed

summary(AVG_AMT_PER_CHQ_TXN)
hist(AVG_AMT_PER_CHQ_TXN)
boxplot(AVG_AMT_PER_CHQ_TXN)
#skewed, falls on the lower side

summary(AVG_AMT_PER_NET_TXN)
hist(AVG_AMT_PER_NET_TXN)
boxplot(AVG_AMT_PER_NET_TXN)
#skewed, Most data falls on the lower side
```

```
summary(AVG_AMT_PER_MOB_TXN)
hist(AVG_AMT_PER_MOB_TXN)
boxplot(AVG_AMT_PER_MOB_TXN)
#Right-skewed, falls on the lower side

summary(FLG_HAS_NOMINEE)
hist(FLG_HAS_NOMINEE)
boxplot(FLG_HAS_NOMINEE)
#Most of the customers have nominee
chisq.test(TARGET, as.factor(FLG_HAS_NOMINEE))
#Not significant neglect - [38]

summary(FLG_HAS_OLD_LOAN)
hist(FLG_HAS_OLD_LOAN)
boxplot(FLG_HAS_OLD_LOAN)
chisq.test(TARGET, as.factor(FLG_HAS_OLD_LOAN))
#Eventhough not Significant, considering this feature as it might be useful
in assesing old loan dependency

summary(random)
hist(random)
boxplot(random)
#random is a continuous variable in increasing order and will be used for
creating development and testing sample
#neglect - [40]

#Dimension Reduction from the EDA (Filter Method Approach)
PL_data <- subset(PL_data, select = -c(CUST_ID))
PL_data <- subset(PL_data, select = -c(AGE_BKT))
PL_data <- subset(PL_data, select = -c(ACC_OP_DATE))
PL_data <- subset(PL_data, select = -c(AMT_OTH_BK_ATM_USG_CHGS))
PL_data <- subset(PL_data, select = -c(AMT_MIN_BAL_NMC_CHGS))
PL_data <- subset(PL_data, select = -c(NO_OF_IW_CHQ_BNC_TXNS))
PL_data <- subset(PL_data, select = -c(NO_OF_OW_CHQ_BNC_TXNS))
PL_data <- subset(PL_data, select = -c(FLG_HAS_NOMINEE))

#dim(PL_data)

#Validating with wrapper method
#install.packages('Boruta')
library(Boruta)

#Feature Selection (Wrapper Method)
set.seed(123)
boruta.train <- Boruta(TARGET~. ,data=PL_data, doTrace = 2)

print(boruta.train)
#Boruta method has confirmed all the features to be important except for
random
```

```
##CART MOdel

#Installation of Necessary Packages
#install.packages('rpart.plot')
#install.packages('rattle')
#install.packages('caret')
#install.packages('ROCR')
#install.packages('ineq')

#loading the libraries
library(rpart)
library(rpart.plot)
library(rattle)
library(RColorBrewer)
library(caret)
library(ROCR)
library(ineq)

#Asssigning the data
PL_CART_data <- PL_data

#dim(PL_CART_data)

#Creation of development and test sample
#Splitting the data into dev(70%) and testing(30%) sample based on the
random number
PL_CART_data.dev <- PL_CART_data[which(PL_CART_data$random <= 0.7),]
PL_CART_data.test <- PL_CART_data[which(PL_CART_data$random > 0.7),]

##Viewing the Development sample
PL_CART_data.dev <- subset(PL_CART_data.dev, select = -c(random))
#head(PL_CART_data.dev)

dim(PL_CART_data.dev)
attach(PL_CART_data.dev)

#view the Testing sample
PL_CART_data.test <- subset(PL_CART_data.test, select = -c(random))

#head(PL_CART_data.test)
dim(PL_CART_data.test)

#check to see if the development and testing sample are partitioned
correctly
table(PL_CART_data.dev$TARGET)
table(PL_CART_data.test$TARGET)

prop.table(table(PL_CART_data.dev$TARGET))
prop.table(table(PL_CART_data.test$TARGET))
```

```
#both have target, approximately of the same proportion


#Defining control parameters
r.ctrl = rpart.control(minsplit=100, minbucket = 30, cp = 0, xval = 10)
#Using rpart to build the tree
PL_CART_data.tree <- rpart(formula = TARGET ~ ., data = PL_CART_data.dev,
method = "class", control = r.ctrl)
PL_CART_data.tree
fancyRpartPlot(PL_CART_data.tree)

##To see how the tree performs
printcp(PL_CART_data.tree)
plotcp(PL_CART_data.tree)

##Pruning the tree
PL_CART_data.ptree<- prune(PL_CART_data.tree, cp= 0.0022  ,"CP")
printcp(PL_CART_data.ptree)
fancyRpartPlot(PL_CART_data.ptree, uniform=TRUE,  main="Pruned
Classification Tree")

#Predicting training data
##Scoring
PL_CART_data.dev$predict.class <- predict(PL_CART_data.ptree,
PL_CART_data.dev, type="class")
PL_CART_data.dev$predict.score <- predict(PL_CART_data.ptree,
PL_CART_data.dev)
#View(PL_CART_data.dev)
head(PL_CART_data.dev)

##Model performance
#Rank ordering
#Deciling
decile <- function(x){
  deciles <- vector(length=10)
  for (i in seq(0.1,1,.1)){
    deciles[i*10] <- quantile(x, i, na.rm=T)
  }
  return (
    ifelse(x<deciles[1], 1,
           ifelse(x<deciles[2], 2,
                  ifelse(x<deciles[3], 3,
                         ifelse(x<deciles[4], 4,
                                ifelse(x<deciles[5], 5,
                                       ifelse(x<deciles[6], 6,
                                              ifelse(x<deciles[7], 7,
                                                     ifelse(x<deciles[8],
8,

ifelse(x<deciles[9], 9, 10
```

```
                                                              ))))))))))
}

#Assigning deciles to the data
PL_CART_data.dev$deciles <- decile(PL_CART_data.dev$predict.score[,2])

#view(PL_CART_data.dev)
head(PL_CART_data.dev)

##Ranking the data
library(data.table)
#Creating rank table
tmp_DT = data.table(PL_CART_data.dev)
rank <- tmp_DT[, list(
  cnt = length(TARGET),
  cnt_resp = sum(TARGET),
  cnt_non_resp = sum(TARGET == 0)) ,
  by=deciles][order(-deciles)]
rank$rrate <- round(rank$cnt_resp * 100 / rank$cnt,2);
rank$cum_resp <- cumsum(rank$cnt_resp)
rank$cum_non_resp <- cumsum(rank$cnt_non_resp)
rank$cum_perct_resp <- round(rank$cum_resp * 100 / sum(rank$cnt_resp),2);
rank$cum_perct_non_resp <- round(rank$cum_non_resp * 100 /
sum(rank$cnt_non_resp),2);
rank$ks <- abs(rank$cum_perct_resp - rank$cum_perct_non_resp);
rank

pred <- prediction(PL_CART_data.dev$predict.score[,2],
PL_CART_data.dev$TARGET)
perf <- performance(pred, "tpr", "fpr")
#plot(perf)
KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])
auc <- performance(pred,"auc");
auc <- as.numeric(auc@y.values)

gini = ineq(PL_CART_data.dev$predict.score[,2], type="Gini")

with(PL_CART_data.dev, table(TARGET, predict.class))
auc
KS
gini

#predicting test data
#Scoring the holdout sample
PL_CART_data.test$predict.class <- predict(PL_CART_data.ptree,
PL_CART_data.test, type="class")
PL_CART_data.test$predict.score <- predict(PL_CART_data.ptree,
PL_CART_data.test)

#head(PL_CART_data.test)
```

```
PL_CART_data.test$deciles <- decile(PL_CART_data.test$predict.score[,2])

#head(PL_CART_data.test)

tmp_DT2 = data.table(PL_CART_data.test)
rank2 <- tmp_DT2[, list(
  cnt = length(TARGET),
  cnt_resp = sum(TARGET),
  cnt_non_resp = sum(TARGET == 0)) ,
  by=deciles][order(-deciles)]
rank2$rrate <- round(rank2$cnt_resp * 100 / rank2$cnt,2);
rank2$cum_resp <- cumsum(rank2$cnt_resp)
rank2$cum_non_resp <- cumsum(rank2$cnt_non_resp)
rank2$cum_perct_resp <- round(rank2$cum_resp * 100 /
sum(rank2$cnt_resp),2);
rank2$cum_perct_non_resp <- round(rank2$cum_non_resp * 100 /
sum(rank2$cnt_non_resp),2);
rank2$ks <- abs(rank2$cum_perct_resp - rank2$cum_perct_non_resp);
rank2

pred <- prediction(PL_CART_data.test$predict.score[,2],
PL_CART_data.test$TARGET)
perf <- performance(pred, "tpr", "fpr")
#plot(perf)

KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])
auc <- performance(pred,"auc");
auc <- as.numeric(auc@y.values)

gini = ineq(PL_CART_data.test$predict.score[,2], type="Gini")

with(PL_CART_data.test, table(TARGET, predict.class))
auc
KS
gini


#cART Model on Balanced data
#install.packages('ROSE')
library(ROSE)

PL_CART_data <- PL_data

PL_CART_data.dev <- PL_CART_data[which(PL_CART_data$random <= 0.7),]
PL_CART_data.test <- PL_CART_data[which(PL_CART_data$random > 0.7),]

PL_CART_data.dev <- subset(PL_CART_data.dev, select = -c(random))

#creating oversampled data
```

```
PL_CART_data.dev.over <-
ovun.sample(TARGET~.,data=PL_CART_data.dev,method="over", N=12141*2)$data
table(PL_CART_data.dev.over$TARGET)

dim(PL_CART_data.dev.over)
#head(PL_CART_data.dev.over)

#control parameter
r.ctrl = rpart.control(minsplit=100,minbucket = 30,cp = 0,xval = 10)

PL_CART_data.dev.over.tree <- rpart(formula = TARGET ~ .,data =
PL_CART_data.dev.over,method = "class",control = r.ctrl)
#PL_CART_data.dev.over.tree
fancyRpartPlot(PL_CART_data.dev.over.tree)

##To see how the tree performs
printcp(PL_CART_data.dev.over.tree)
plotcp(PL_CART_data.dev.over.tree)

##Pruning the tree
PL_CART_data.dev.over.ptree<- prune(PL_CART_data.dev.over.tree, cp= 0.00030
,"CP")
printcp(PL_CART_data.dev.over.ptree)
fancyRpartPlot(PL_CART_data.dev.over.ptree, uniform=TRUE)

#Predicting training data
##Scoring
PL_CART_data.dev.over$predict.class <- predict(PL_CART_data.dev.over.ptree,
PL_CART_data.dev.over, type="class")
PL_CART_data.dev.over$predict.score <- predict(PL_CART_data.dev.over.ptree,
PL_CART_data.dev.over)
#View(PL_CART_data.dev)
#head(PL_CART_data.dev.over)

##Model performance
#Rank ordering
#Deciling
decile <- function(x){
  deciles <- vector(length=10)
  for (i in seq(0.1,1,.1)){
    deciles[i*10] <- quantile(x, i, na.rm=T)
  }
  return (
    ifelse(x<deciles[1], 1,
           ifelse(x<deciles[2], 2,
                  ifelse(x<deciles[3], 3,
                         ifelse(x<deciles[4], 4,
                                ifelse(x<deciles[5], 5,
                                       ifelse(x<deciles[6], 6,
                                              ifelse(x<deciles[7], 7,
```

```
                                                        ifelse(x<deciles[8],
8,

ifelse(x<deciles[9], 9, 10
                                                                ))))))))))
}

#Assigning deciles to the data
PL_CART_data.dev.over$deciles <-
decile(PL_CART_data.dev.over$predict.score[,2])

#view(PL_CART_data.dev)
#head(PL_CART_data.dev.over)

##Ranking the data
library(data.table)
#Creating rank table
tmp_DT_BT = data.table(PL_CART_data.dev.over)
rank <- tmp_DT_BT[, list(
  cnt = length(TARGET),
  cnt_resp = sum(TARGET),
  cnt_non_resp = sum(TARGET == 0)) ,
  by=deciles][order(-deciles)]
rank$rrate <- round(rank$cnt_resp * 100 / rank$cnt,2);
rank$cum_resp <- cumsum(rank$cnt_resp)
rank$cum_non_resp <- cumsum(rank$cnt_non_resp)
rank$cum_perct_resp <- round(rank$cum_resp * 100 / sum(rank$cnt_resp),2);
rank$cum_perct_non_resp <- round(rank$cum_non_resp * 100 /
sum(rank$cnt_non_resp),2);
rank$ks <- abs(rank$cum_perct_resp - rank$cum_perct_non_resp);
rank

pred <- prediction(PL_CART_data.dev.over$predict.score[,2],
PL_CART_data.dev.over$TARGET)
perf <- performance(pred, "tpr", "fpr")
#plot(perf)
KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])
auc <- performance(pred,"auc");
auc <- as.numeric(auc@y.values)

gini = ineq(PL_CART_data.dev.over$predict.score[,2], type="Gini")

with(PL_CART_data.dev.over, table(TARGET, predict.class))
auc
KS
gini

#predicting test data
#Scoring the holdout sample
PL_CART_data.test.new <- PL_CART_data.test
```

```
PL_CART_data.test.new <- subset(PL_CART_data.test.new, select = -c(random))

dim(PL_CART_data.test.new)

PL_CART_data.test.new$predict.class <- predict(PL_CART_data.dev.over.ptree,
PL_CART_data.test.new, type="class")
PL_CART_data.test.new$predict.score <- predict(PL_CART_data.dev.over.ptree,
PL_CART_data.test.new)

#head(PL_CART_data.test.new)

decile <- function(x){
  deciles <- vector(length=10)
  for (i in seq(0.1,1,.1)){
    deciles[i*10] <- quantile(x, i, na.rm=T)
  }
  return (
    ifelse(x<deciles[1], 1,
           ifelse(x<deciles[2], 2,
                  ifelse(x<deciles[3], 3,
                         ifelse(x<deciles[4], 4,
                                ifelse(x<deciles[5], 5,
                                       ifelse(x<deciles[6], 6,
                                              ifelse(x<deciles[7], 7,
                                                     ifelse(x<deciles[8],
8,

ifelse(x<deciles[9], 9, 10
                                                            ))))))))))
}

PL_CART_data.test.new$deciles <-
decile(PL_CART_data.test.new$predict.score[,2])

#head(PL_CART_data.test.new)

tmp_DT3 = data.table(PL_CART_data.test.new)
rank2 <- tmp_DT3[, list(
  cnt = length(TARGET),
  cnt_resp = sum(TARGET),
  cnt_non_resp = sum(TARGET == 0)) ,
  by=deciles][order(-deciles)]
rank2$rrate <- round(rank2$cnt_resp * 100 / rank2$cnt,2);
rank2$cum_resp <- cumsum(rank2$cnt_resp)
rank2$cum_non_resp <- cumsum(rank2$cnt_non_resp)
rank2$cum_perct_resp <- round(rank2$cum_resp * 100 /
sum(rank2$cnt_resp),2);
rank2$cum_perct_non_resp <- round(rank2$cum_non_resp * 100 /
sum(rank2$cnt_non_resp),2);
rank2$ks <- abs(rank2$cum_perct_resp - rank2$cum_perct_non_resp);
```

```
rank2

pred <- prediction(PL_CART_data.test.new$predict.score[,2],
PL_CART_data.test.new$TARGET)
perf <- performance(pred, "tpr", "fpr")
#plot(perf)
KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])
auc <- performance(pred,"auc");
auc <- as.numeric(auc@y.values)

gini = ineq(PL_CART_data.test.new$predict.score[,2], type="Gini")

with(PL_CART_data.test.new, table(TARGET, predict.class))
auc
KS
gini



# RandomForest Model
PL_RF_data <- PL_data

#Creation of development and test sample
#Splitting the data into dev(70%) and testing(30%) sample based on the
random number
PL_RF_data.dev <- PL_RF_data[which(PL_RF_data$random <= 0.7),]
PL_RF_data.test <- PL_RF_data[which(PL_RF_data$random > 0.7),]

##Viewing the Development sample
#head(PL_RF_data.dev)
dim(PL_RF_data.dev)

PL_RF_data.dev <- subset(PL_RF_data.dev, select = -c(random))


#view the Testing sample
PL_RF_data.test <- subset(PL_RF_data.test, select = -c(random))

#head(PL_RF_data.test)
dim(PL_RF_data.test)

##Creating random forest
#install.packages("randomForest")
#ntree: number of trees to grow
#mtry: number of variables to be considered for split (sqrt(no of
features))
#nodesize: minimum size of terminal nodes (2-3% of dataset)
library(randomForest)
set.seed(123)
PL.RF <- randomForest(as.factor(TARGET) ~ ., data = PL_RF_data.dev,
```

```
                      ntree=501, mtry = 5, nodesize = 250,
                      importance=TRUE)
print(PL.RF)

#To choose optimum value of ntree
plot(PL.RF, main="")
legend("topright", c("OOB", "0", "1"), text.col=1:6, lty=1:3, col=1:3)
title(main="Error Rates")

PL.RF$err.rate
#choose ntree=121

#List the iimportance of the variable
impVar <- round(randomForest::importance(PL.RF), 2)
impVar[order(impVar[,3], decreasing=TRUE),]

#Tuning Random Forest
tRF <- tuneRF(x = PL_RF_data.dev[,-c(1)],
              y=as.factor(PL_RF_data.dev$TARGET),
              mtryStart = 5,
              ntreeTry=121,
              stepFactor = 1.5,
              improve = 0.0001,
              trace=TRUE,
              plot = TRUE,
              doBest = TRUE,
              nodesize = 250,
              importance=TRUE
)

PL.RF <- randomForest(as.factor(TARGET) ~ ., data = PL_RF_data.dev,
                      ntree=121, mtry = 22, nodesize = 250,
                      importance=TRUE)
print(PL.RF)

## Scoring syntax
PL_RF_data.dev$predict.class <- predict(PL.RF, PL_RF_data.dev,
type="class")
PL_RF_data.dev$predict.score <- predict(PL.RF, PL_RF_data.dev, type="prob")

## deciling code
decile <- function(x){
  deciles <- vector(length=10)
  for (i in seq(0.1,1,.1)){
    deciles[i*10] <- quantile(x, i, na.rm=T)
  }
  return (
    ifelse(x<deciles[1], 1,
           ifelse(x<deciles[2], 2,
                  ifelse(x<deciles[3], 3,
```

```
                         ifelse(x<deciles[4], 4,
                                ifelse(x<deciles[5], 5,
                                       ifelse(x<deciles[6], 6,
                                              ifelse(x<deciles[7], 7,
                                                     ifelse(x<deciles[8],
8,

ifelse(x<deciles[9], 9, 10
                                                            ))))))))))
}

## deciling
PL_RF_data.dev$deciles <- decile(PL_RF_data.dev$predict.score[,2])


## Ranking code
library(data.table)
tmp_DT = data.table(PL_RF_data.dev)
rank <- tmp_DT[, list(
  cnt = length(TARGET),
  cnt_resp = sum(TARGET),
  cnt_non_resp = sum(TARGET == 0)) ,
  by=deciles][order(-deciles)]
rank$rrate <- round(rank$cnt_resp * 100 / rank$cnt,2);
rank$cum_resp <- cumsum(rank$cnt_resp)
rank$cum_non_resp <- cumsum(rank$cnt_non_resp)
rank$cum_rel_resp <- round(rank$cum_resp / sum(rank$cnt_resp),2);
rank$cum_rel_non_resp <- round(rank$cum_non_resp /
sum(rank$cnt_non_resp),2);
rank$ks <- abs(rank$cum_rel_resp - rank$cum_rel_non_resp);
rank

pred <- prediction(PL_RF_data.dev$predict.score[,2], PL_RF_data.dev$TARGET)
perf <- performance(pred, "tpr", "fpr")
#plot(perf)
KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])
auc <- performance(pred,"auc");
auc <- as.numeric(auc@y.values)

gini = ineq(PL_RF_data.dev$predict.score[,2], type="Gini")

with(PL_RF_data.dev, table(TARGET, predict.class))
auc
KS
gini

#validation on testing sample
PL_RF_data.test$predict.class <- predict(PL.RF, PL_RF_data.test,
type="class")
```

```
PL_RF_data.test$predict.score <- predict(PL.RF, PL_RF_data.test,
type="prob")

## deciling code
decile <- function(x){
  deciles <- vector(length=10)
  for (i in seq(0.1,1,.1)){
    deciles[i*10] <- quantile(x, i, na.rm=T)
  }
  return (
    ifelse(x<deciles[1], 1,
           ifelse(x<deciles[2], 2,
                  ifelse(x<deciles[3], 3,
                         ifelse(x<deciles[4], 4,
                                ifelse(x<deciles[5], 5,
                                       ifelse(x<deciles[6], 6,
                                              ifelse(x<deciles[7], 7,
                                                     ifelse(x<deciles[8],
8,

ifelse(x<deciles[9], 9, 10
                                                                   ))))))))))
}

## deciling
PL_RF_data.test$deciles <- decile(PL_RF_data.test$predict.score[,2])


## Ranking code
library(data.table)
tmp_DT = data.table(PL_RF_data.test)
rank <- tmp_DT[, list(
  cnt = length(TARGET),
  cnt_resp = sum(TARGET),
  cnt_non_resp = sum(TARGET == 0)) ,
  by=deciles][order(-deciles)]
rank$rrate <- round(rank$cnt_resp * 100 / rank$cnt,2);
rank$cum_resp <- cumsum(rank$cnt_resp)
rank$cum_non_resp <- cumsum(rank$cnt_non_resp)
rank$cum_rel_resp <- round(rank$cum_resp / sum(rank$cnt_resp),2);
rank$cum_rel_non_resp <- round(rank$cum_non_resp /
sum(rank$cnt_non_resp),2);
rank$ks <- abs(rank$cum_rel_resp - rank$cum_rel_non_resp);
rank

pred <- prediction(PL_RF_data.test$predict.score[,2],
PL_RF_data.test$TARGET)
perf <- performance(pred, "tpr", "fpr")
plot(perf)
KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])
```

```
auc <- performance(pred,"auc");
auc <- as.numeric(auc@y.values)

gini = ineq(PL_RF_data.test$predict.score[,2], type="Gini")

with(PL_RF_data.test, table(TARGET, predict.class))
auc
KS
gini


#NN Model

PL_NN_data <- PL_data

#Creation of development and test sample
#Splitting the data into dev(70%) and testing(30%) sample based on the
random number
PL_NN_data.dev <- PL_NN_data[which(PL_NN_data$random <= 0.7),]
PL_NN_data.test <- PL_NN_data[which(PL_NN_data$random > 0.7),]

##Viewing the Development sample
#head(PL_NN_data.dev)
dim(PL_NN_data.dev)

PL_NN_data.dev <- subset(PL_NN_data.dev, select = -c(random))


#view the Testing sample
PL_NN_data.test <- subset(PL_NN_data.test, select = -c(random))

#head(PL_RF_data.test)
dim(PL_NN_data.test)

##Converting Categorical Variables into dummy variables
# Gender
GEN.matrix <- model.matrix(~ GENDER - 1, data = PL_NN_data.dev)
PL_NN_data.dev <- data.frame(PL_NN_data.dev, GEN.matrix)

GEN.matrix <- model.matrix(~ GENDER - 1, data = PL_NN_data.test)
PL_NN_data.test <- data.frame(PL_NN_data.test, GEN.matrix)

# Occupation
occ.matrix <- model.matrix(~ OCCUPATION - 1, data = PL_NN_data.dev)
PL_NN_data.dev <- data.frame(PL_NN_data.dev, occ.matrix)

occ.matrix <- model.matrix(~ OCCUPATION - 1, data = PL_NN_data.test)
PL_NN_data.test <- data.frame(PL_NN_data.test, occ.matrix)

# ACC_TYPE
```

```
ACCTYP.matrix <- model.matrix(~ ACC_TYPE - 1, data = PL_NN_data.dev)
PL_NN_data.dev <- data.frame(PL_NN_data.dev, ACCTYP.matrix)

ACCTYP.matrix <- model.matrix(~ ACC_TYPE - 1, data = PL_NN_data.test)
PL_NN_data.test <- data.frame(PL_NN_data.test, ACCTYP.matrix)

dim(PL_NN_data.dev)
#head(PL_NN_data.dev)

dim(PL_NN_data.test)
#head(PL_NN_data.test)

## Response Rate
sum(PL_NN_data.dev$TARGET) / nrow(PL_NN_data.dev)
sum(PL_NN_data.test$TARGET) / nrow(PL_NN_data.test)

##Installing the Neural Net package
#install.packages("neuralnet")
library(neuralnet)

##Scaling variables
x <- subset(PL_NN_data.dev,
                     select = c("AGE", "BALANCE", "SCR", "HOLDING_PERIOD",
"LEN_OF_RLTN_IN_MNTH",
                                "NO_OF_L_CR_TXNS", "NO_OF_L_DR_TXNS",
"TOT_NO_OF_L_TXNS",
                                "NO_OF_BR_CSH_WDL_DR_TXNS",
"NO_OF_ATM_DR_TXNS",
                                "NO_OF_NET_DR_TXNS", "NO_OF_MOB_DR_TXNS",
"NO_OF_CHQ_DR_TXNS",
                                "FLG_HAS_CC", "AMT_ATM_DR",
"AMT_BR_CSH_WDL_DR", "AMT_CHQ_DR",
                                "AMT_NET_DR", "AMT_MOB_DR", "AMT_L_DR",
                                "FLG_HAS_ANY_CHGS", "AVG_AMT_PER_ATM_TXN",
"AVG_AMT_PER_CSH_WDL_TXN",
                                "AVG_AMT_PER_CHQ_TXN",
"AVG_AMT_PER_NET_TXN", "AVG_AMT_PER_MOB_TXN",
                                "FLG_HAS_OLD_LOAN", "GENDERF", "GENDERM",
"GENDERO",
                                "OCCUPATIONPROF", "OCCUPATIONSAL",
"OCCUPATIONSELF.EMP",
                                "OCCUPATIONSENP", "ACC_TYPECA",
"ACC_TYPESA")
)


PL_NN_data.dev.scaled <- scale(x)
PL_NN_data.dev.scaled <- cbind(PL_NN_data.dev[1], PL_NN_data.dev.scaled)

#View(PL_NN_data.dev.scaled)
```

```
dim(PL_NN_data.dev.scaled)

y <- subset(PL_NN_data.test,
            select = c("AGE", "BALANCE", "SCR", "HOLDING_PERIOD",
"LEN_OF_RLTN_IN_MNTH",
                       "NO_OF_L_CR_TXNS", "NO_OF_L_DR_TXNS",
"TOT_NO_OF_L_TXNS",
                       "NO_OF_BR_CSH_WDL_DR_TXNS", "NO_OF_ATM_DR_TXNS",
                       "NO_OF_NET_DR_TXNS", "NO_OF_MOB_DR_TXNS",
"NO_OF_CHQ_DR_TXNS",
                       "FLG_HAS_CC", "AMT_ATM_DR", "AMT_BR_CSH_WDL_DR",
"AMT_CHQ_DR",
                       "AMT_NET_DR", "AMT_MOB_DR", "AMT_L_DR",
                       "FLG_HAS_ANY_CHGS", "AVG_AMT_PER_ATM_TXN",
"AVG_AMT_PER_CSH_WDL_TXN",
                       "AVG_AMT_PER_CHQ_TXN", "AVG_AMT_PER_NET_TXN",
"AVG_AMT_PER_MOB_TXN",
                       "FLG_HAS_OLD_LOAN", "GENDERF", "GENDERM", "GENDERO",
                       "OCCUPATIONPROF", "OCCUPATIONSAL",
"OCCUPATIONSELF.EMP",
                       "OCCUPATIONSENP", "ACC_TYPECA", "ACC_TYPESA")
)


PL_NN_data.test.scaled <- scale(y)
PL_NN_data.test.scaled <- cbind(PL_NN_data.test[1], PL_NN_data.test.scaled)

#View(PL_NN_data.test.scaled)
dim(PL_NN_data.test.scaled)


##Creating a neural network
nn <- neuralnet(formula = TARGET ~
                  AGE +
                  BALANCE +
                  SCR +
                  HOLDING_PERIOD +
                  LEN_OF_RLTN_IN_MNTH +
                  NO_OF_L_CR_TXNS +
                  NO_OF_L_DR_TXNS +
                  TOT_NO_OF_L_TXNS +
                  NO_OF_BR_CSH_WDL_DR_TXNS +
                  NO_OF_ATM_DR_TXNS +
                  NO_OF_NET_DR_TXNS +
                  NO_OF_MOB_DR_TXNS +
                  NO_OF_CHQ_DR_TXNS +
                  FLG_HAS_CC +
                  AMT_ATM_DR +
                  AMT_BR_CSH_WDL_DR +
                  AMT_CHQ_DR +
```

```
                  AMT_NET_DR +
                  AMT_MOB_DR +
                  AMT_L_DR +
                  FLG_HAS_ANY_CHGS +
                  AVG_AMT_PER_ATM_TXN +
                  AVG_AMT_PER_CSH_WDL_TXN +
                  AVG_AMT_PER_CHQ_TXN +
                  AVG_AMT_PER_NET_TXN +
                  AVG_AMT_PER_MOB_TXN +
                  FLG_HAS_OLD_LOAN +
                  GENDERF +
                  GENDERM +
                  GENDERO +
                  OCCUPATIONPROF +
                  OCCUPATIONSAL +
                  OCCUPATIONSELF.EMP +
                  OCCUPATIONSENP +
                  ACC_TYPECA +
                  ACC_TYPESA ,
                data = PL_NN_data.dev.scaled,
                hidden = 3,
                err.fct = "sse",
                linear.output = FALSE,
                lifesign = "full",
                lifesign.step = 10,
                threshold = 0.1,
                stepmax = 2000
)


plot (nn)


## Assigning the Probabilities to Dev Sample
PL_NN_data.dev.scaled$Prob = nn$net.result[[1]]

## The distribution of the estimated probabilities
quantile(PL_NN_data.dev.scaled$Prob,
c(0,1,5,10,25,50,75,90,95,98,99,100)/100)
hist(PL_NN_data.dev.scaled$Prob)


## deciling code
decile <- function(x){
  deciles <- vector(length=10)
  for (i in seq(0.1,1,.1)){
    deciles[i*10] <- quantile(x, i, na.rm=T)
  }
  return (
    ifelse(x<deciles[1], 1,
```

```
            ifelse(x<deciles[2], 2,
                   ifelse(x<deciles[3], 3,
                          ifelse(x<deciles[4], 4,
                                 ifelse(x<deciles[5], 5,
                                        ifelse(x<deciles[6], 6,
                                               ifelse(x<deciles[7], 7,
                                                      ifelse(x<deciles[8],
8,

ifelse(x<deciles[9], 9, 10
                                                             ))))))))))
}

## deciling
PL_NN_data.dev.scaled$deciles <- decile(PL_NN_data.dev.scaled$Prob)

## Ranking code
##install.packages("data.table")
library(data.table)
library(scales)

tmp_DT = data.table(PL_NN_data.dev.scaled)
rank <- tmp_DT[, list(
  cnt = length(TARGET),
  cnt_resp = sum(TARGET),
  cnt_non_resp = sum(TARGET == 0)) ,
  by=deciles][order(-deciles)]
rank$rrate <- round (rank$cnt_resp / rank$cnt,2);
rank$cum_resp <- cumsum(rank$cnt_resp)
rank$cum_non_resp <- cumsum(rank$cnt_non_resp)
rank$cum_rel_resp <- round(rank$cum_resp / sum(rank$cnt_resp),2);
rank$cum_rel_non_resp <- round(rank$cum_non_resp /
sum(rank$cnt_non_resp),2);
rank$ks <- abs(rank$cum_rel_resp - rank$cum_rel_non_resp);
rank$rrate <- percent(rank$rrate)
rank$cum_rel_resp <- percent(rank$cum_rel_resp)
rank$cum_rel_non_resp <- percent(rank$cum_rel_non_resp)

rank

PL_NN_data.dev.scaled$Class = ifelse(PL_NN_data.dev.scaled$Prob>0.5,1,0)

with( PL_NN_data.dev.scaled, table(TARGET, as.factor(Class) ))

pred <- prediction(PL_NN_data.dev.scaled$Prob,
PL_NN_data.dev.scaled$TARGET)
perf <- performance(pred, "tpr", "fpr")
plot(perf)

KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])
```

```
KS

auc <- performance(pred,"auc");
auc <- as.numeric(auc@y.values)
auc

gini = ineq(PL_NN_data.dev.scaled$Prob, type="Gini")

gini


#validating on test Sample
compute.output <- compute(nn, PL_NN_data.test.scaled)
PL_NN_data.test.scaled$Predict.score <- compute.output$net.result

#Assiging probabilities
PL_NN_data.test.scaled$Prob = compute.output$net.result

## deciling
PL_NN_data.test.scaled$deciles <-
decile(PL_NN_data.test.scaled$Predict.score)

## Ranking code
tmp_DT = data.table(PL_NN_data.test.scaled)
rank <- tmp_DT[, list(
  cnt = length(TARGET),
  cnt_resp = sum(TARGET),
  cnt_non_resp = sum(TARGET == 0)) ,
  by=deciles][order(-deciles)]
rank$rrate <- round (rank$cnt_resp / rank$cnt,2);
rank$cum_resp <- cumsum(rank$cnt_resp)
rank$cum_non_resp <- cumsum(rank$cnt_non_resp)
rank$cum_rel_resp <- round(rank$cum_resp / sum(rank$cnt_resp),2);
rank$cum_rel_non_resp <- round(rank$cum_non_resp /
sum(rank$cnt_non_resp),2);
rank$ks <- abs(rank$cum_rel_resp - rank$cum_rel_non_resp);
rank$rrate <- percent(rank$rrate)
rank$cum_rel_resp <- percent(rank$cum_rel_resp)
rank$cum_rel_non_resp <- percent(rank$cum_rel_non_resp)

rank

PL_NN_data.test.scaled$Class = ifelse(PL_NN_data.test.scaled$Prob>0.5,1,0)

with( PL_NN_data.test.scaled, table(TARGET, as.factor(Class) ))

library(ROCR)
detach(package:neuralnet)
pred <- prediction(PL_NN_data.test.scaled$Prob,
PL_NN_data.test.scaled$TARGET)
```

```
perf <- performance(pred, "tpr", "fpr")
#plot(perf)

KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])
KS

auc <- performance(pred,"auc");
auc <- as.numeric(auc@y.values)
auc

gini = ineq(PL_NN_data.test.scaled$Prob, type="Gini")
gini
```