**Churn Reduction**

*Kishore Mohit*

*15 July 2018*

**CONTENT**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Chapter** | **Page No.** |
| 1. | Introduction | 3 |
| 1.1 Problem Statement | 3 |
| 1.2 Data | 3 |
| 2. | Methodology | 5 |
| 2.1 Data Pre-Processing | 5 |
| 2.1.1 Missing Data | 5 |
| 2.1.2 Feature Selection | 7 |
| 2.1.3 Feature Scaling | 13 |
| 2.2 Modeling | 13 |
| 2.2.1 Model Selection | 13 |
| 2.3. Cross Validation | 25 |
| 3. | Conclusion | 27 |
| 3.1 Model Evaluation | 27 |
| 3.2 Model Selection | 31 |
| 3.3 Class Imbalance Problem | 31 |
| 4. | Appendix A – Extra Figures | 35 |
| 5. | Appendix B - Code | 39 |

**CHAPTER 1**

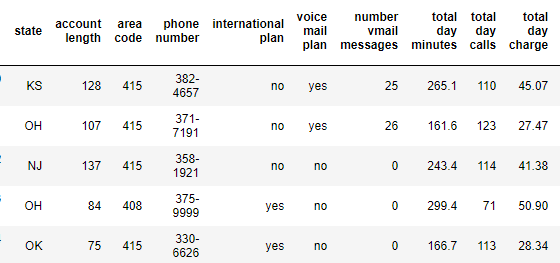
**INTRODUCTION**

* 1. **Problem Statement**

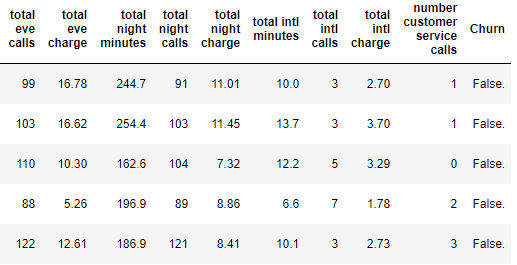
Churn (loss of customers to competition) is a problem for companies because it is more expensive to acquire a new customer than to keep your existing one from leaving. We are provided with a data set in which customer telephone service and call details. This problem statement is targeted at enabling churn reduction using analytics concepts.

* 1. **Data**

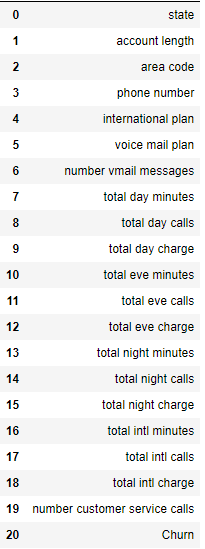
The objective of this Case is to predict customer behavior. We are provided with a public dataset that has customer usage pattern and if the customer has moved or not. We have to develop an algorithm to predict the churn score based on usage pattern.



**Table 1.1 Churn Reduction Train Data (Column: 1-11)**

****

**Table 1.2 Churn Reduction Train Data (Column: 12-21)**

****

**Table 1.3 Predictor Variable**

Our target variable is “Churn” of which we have to Predict and Analysis whether the customer will move or not.

**CHAPTER 2**

**METHODOLOGY**

**2.1 Data Pre-Processing**

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues. Data preprocessing prepares raw data for further processing.

Data goes through a series of steps during preprocessing:

• Data Cleaning: Data is cleansed through processes such as filling in missing values, smoothing the noisy data, or resolving the inconsistencies in the data.

• Data Integration: Data with different representations are put together and conflicts within the data are resolved.

• Data Transformation: Data is normalized, aggregated and generalized.

• Data Reduction: This step aims to present a reduced representation of the data in a data warehouse.

• Data Discretization: Involves the reduction of a number of values of a continuous attribute by dividing the range of attribute intervals.

For Our churn dataset we need following preprocessing before we create a model for the problem and these steps include:

1. Missing value analysis

2. Feature selection

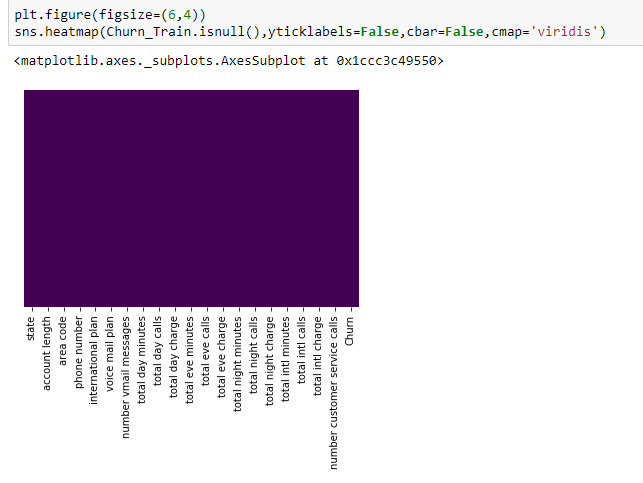
2.1 Correlation Analysis

2.2 Chi square test of Independence

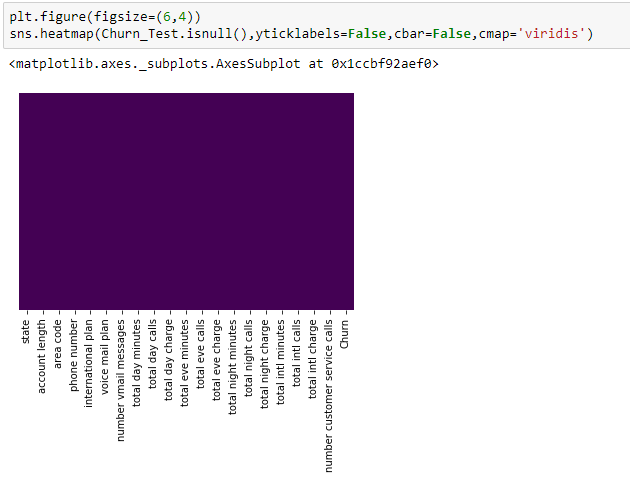
3. Normalization

**2.1.1 Missing Data**

Before starting with cleaning process in preprocessing, we have to impute the missing value using Mean, Median, KNN whichever suits better with the data and have closer to our prediction data. So first we have to analyze missing data in our both data cases i.e. Train and Test Data. As we can visualize in Figure for missing data as below figures for test and train data and there is no missing data in out data set.



**Figure 2.1. Missing data visualize for Train Dataset**

****

**Figure 2.2. Missing data visualize for Test Dataset**

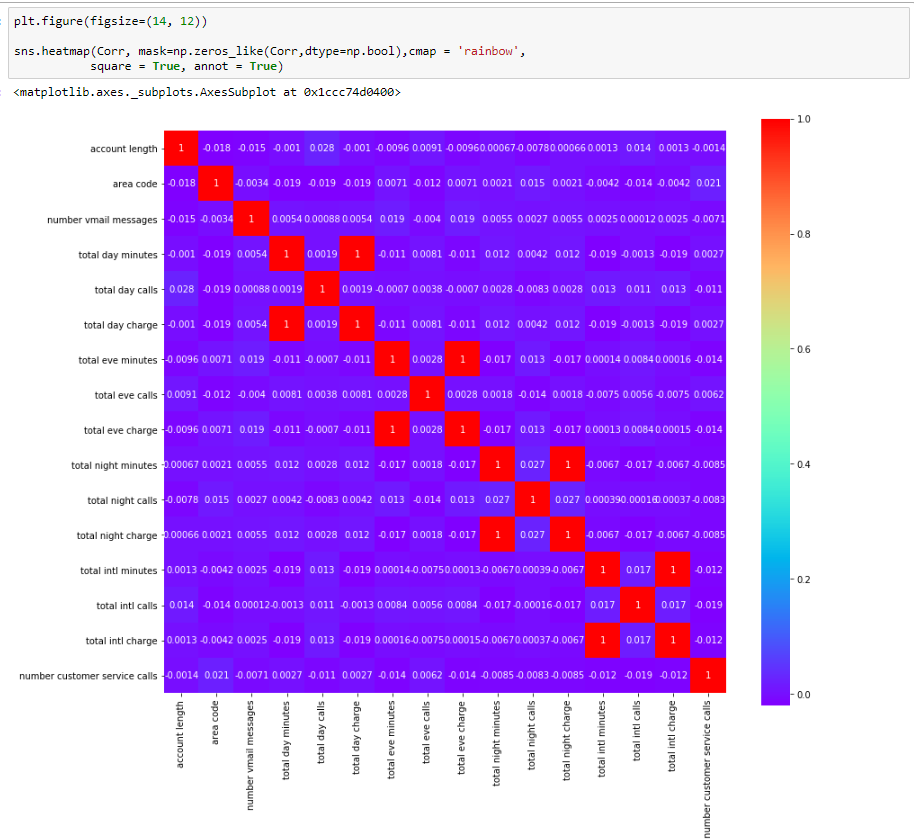
As we can observe in visualize plot that has no missing data where Blue color show the data presence and yellow shows the missing data and we cannot see any missing yellow data. So, we are good to go for next part of problem.

**2.1.2 Feature Selection**

Feature selection is crucial step for any data science model as it provides the insight in the data for analysis and helps to create better model with better accuracy. For our dataset Churn reduction first, we need to analyze continuous variable for the correlation of the variables that requires correlation analysis, that is also known as Pearson correlation analysis.

Correlation is used to test relationships between quantitative variables or categorical variables. In other words, it’s a measure of how things are related. The study of how variables are correlated is called correlation analysis.

**Correlation Analysis**



**Figure 2.3. Correlation Plot on Continuous Variable**

Our correlation plot shows some higher correlation where,

1. Total day minutes and total day charge are very highly correlated.

2. Total eve minutes and total eve charge are very highly correlated.

3. Total night minutes and total night charge are very highly correlated.

4. Total intl minutes and total intl charge are very highly correlated.

Now, after performing correlation and analyzing the correlation matrix we can remove one of the highly correlated variable so that our model can perform well with much accuracy.

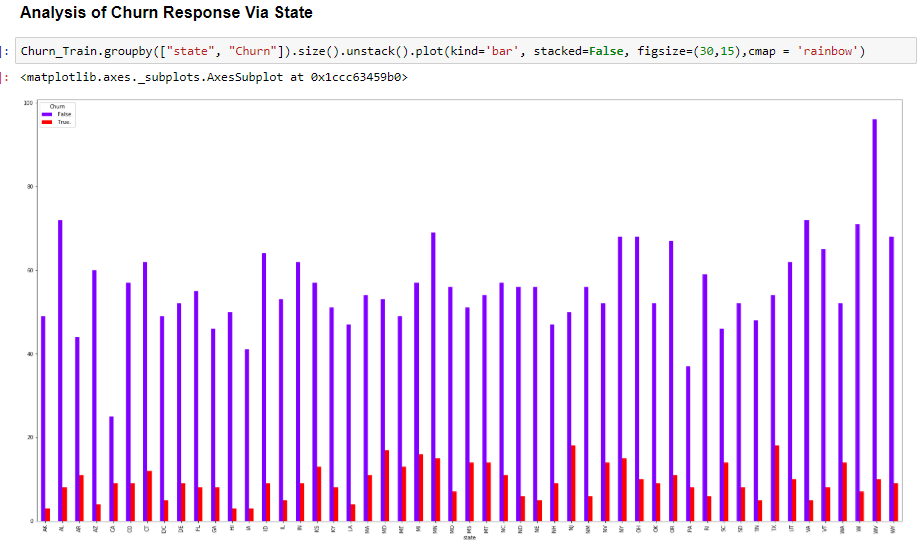
**Chi Square Test of Independence**

After performing correlation for continuous variable, now we need to find most important categorical variable for our model that present in our dataset where chi square test comes into the picture.

A chi-square test for independence compares two variables in a contingency table to see if they are related. In a more general sense, it tests to see whether distributions of [categorical variables](http://www.statisticshowto.com/what-is-a-categorical-variable/) differ from each another.

* A very small chi square test statistic means that your observed data fits your expected data extremely well. In other words, there is a relationship.
* A very large chi square test statistic means that the data does not fit very well. In other words, there isn’t a relationship.

But before performing chi square test, let’s see some visualization for all categorical data relation with our Target variable Churn Reduction, and how other categorical variables are related with Target and what affect it going to make in creation of our model.



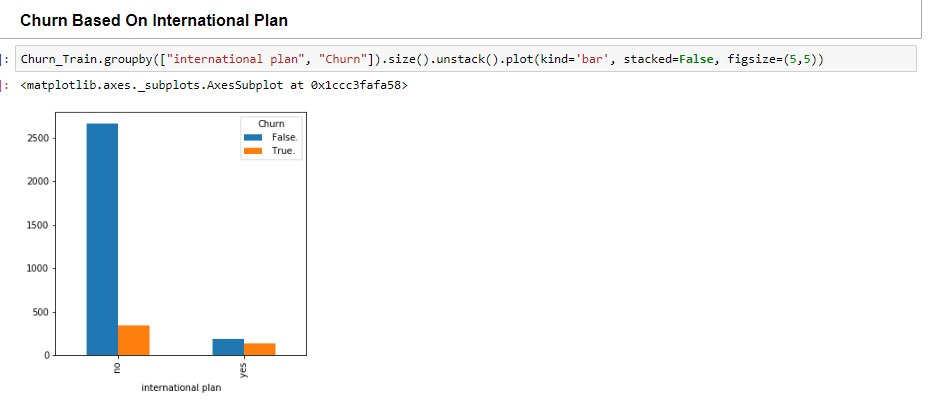
**Figure 2.4. Analysis between Churn and State**



**Figure 2.5. Analysis between Churn and Area Code**

****

**Figure 2.6. Analysis between Churn and Voice Mail Plan**

****

**Figure 2.7. Analysis between Churn and International Plan**

The Chi-Square test of independence is used to evaluate and find out if there is a significant relationship between two categorical variables and frequency of each category for one variable is compared across the categories of the other variable. It shows whether there is a significant association between the two variables and it uses contingency table for better representation and here we can perform chi square test of independence for each of the categorical variable with our target variable to remove the variable that is not dependent with target variable. Scores of chi square test of independence of each categorical variable came out as shown below after performing chi sq. on categorical variables.

[1] "state"

Pearson's Chi-squared test

data: table(Factor\_Data$Churn, Factor\_Data[, i])

X-squared = 83.044, df = 50, p-value = 0.002296

[1] "international.plan"

Pearson's Chi-squared test with Yates' continuity correction

data: table(Factor\_Data$Churn, Factor\_Data[, i])

X-squared = 222.57, df = 1, p-value < 2.2e-16

[1] "voice.mail.plan"

Pearson's Chi-squared test with Yates' continuity correction

data: table(Factor\_Data$Churn, Factor\_Data[, i])

X-squared = 34.132, df = 1, p-value = 5.151e-09

[1] "Churn"

Pearson's Chi-squared test with Yates' continuity correction

data: table(Factor\_Data$Churn, Factor\_Data[, i])

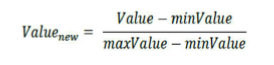
X-squared = 3324.9, df = 1, p-value < 2.2e-16

Here we can observe our p-value for all categorical data is <0.05 and we can say all our categorical data is have 95% significant relation to our target variable and If the p value of the categorical variable is less than 0.05 then we will consider that variable for target variable and can say it is dependent on the categorical variable therefore from both the correlation analysis and chi square test of independence there is some variable that shouldn’t consider for further processing, that are

Numerical: total day minutes, total eve minutes, total night minutes, total intl minutes, phone number.

**2.1.3. Feature Scaling**

For scaling our data, we will be doing normalization to bring our data between 0-1, so it will be well processed during data science process and during model development this is also known as min-max scaling or min-max normalization, that is the simplest method and consists in rescaling the range of features to scale the range in [0, 1] or [−1, 1]. Selecting the target range depends on the nature of the data. The general formula is given as:



We will be performing the normalization on continuous variables that areaccount length, area code, number vmail messages, total day calls, total day charge, total eve calls, total eve charge, total night calls, total night charge, total intl calls, total intl charge, number customer service calls in our dataset. After performing normalization, we are ready for our further model development and analysis.

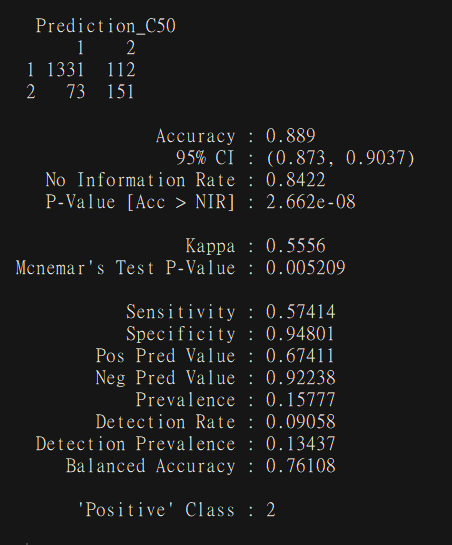
**2.2 Modeling**

**2.2.1 Model selection**

For our model development we will be using 6 classification algorithm and we will select the model based on accuracy, False negative rate, sensitivity and so on.

**Decision Tree**

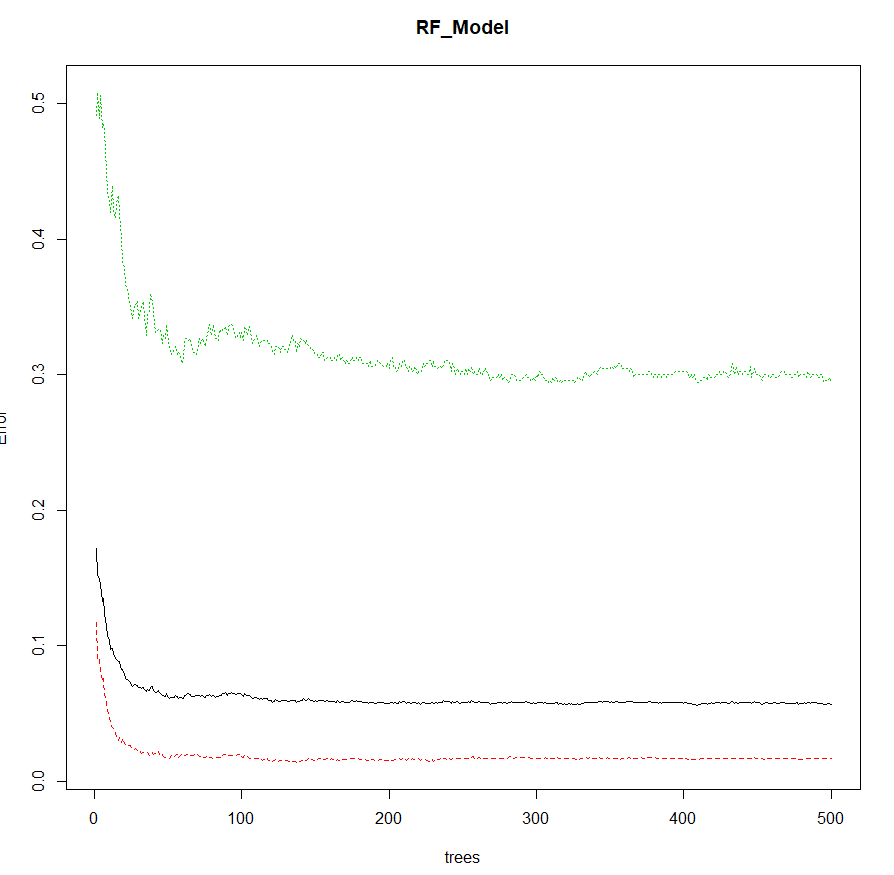
Decision tree is a rule where each branch connects nodes with “and” and multiple branches are determining by “or” and this algorithm can be used for classification and regression. Decision tree is a supervised machine learning algorithm which accept continuous and categorical variables as independent variable. We will be using C5.0 model which is entropy based. The accuracy obtained by Decision tree as given below:

****

**Figure 2.8 Decision Tree Confusion Matrix**

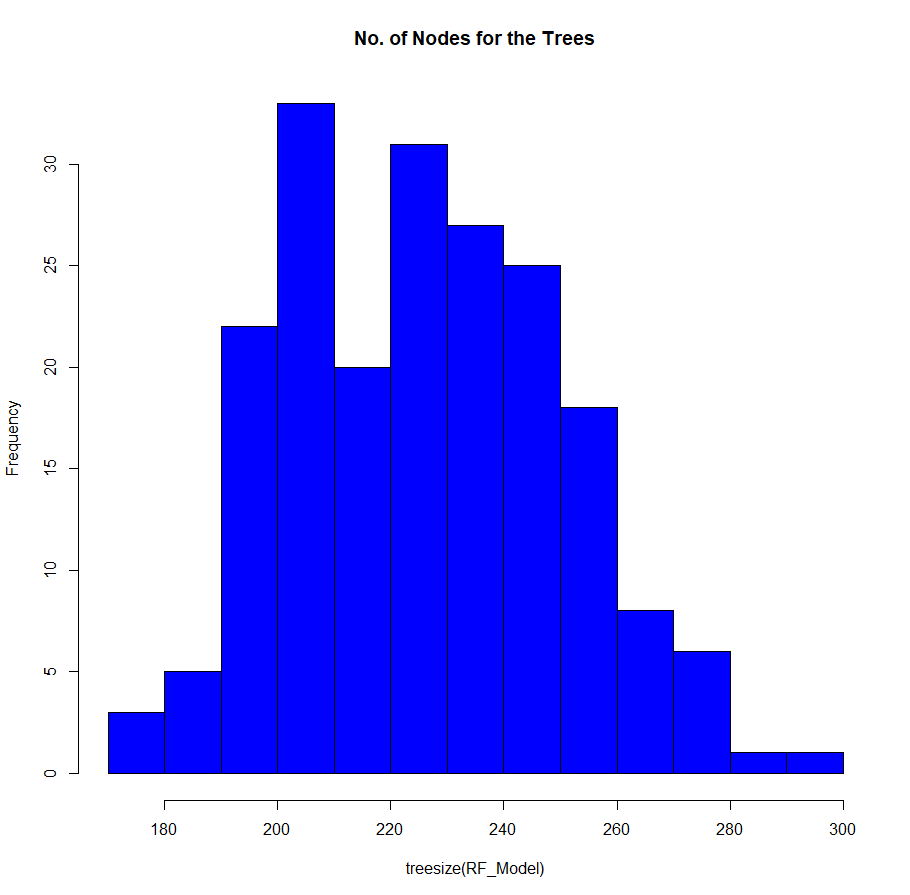
**Random Forest**

Now we will be performing Random forest but some “n” tree for better performance, first we will be taking n=500 and then we will plot the number of tree is being used during our model creation**.**

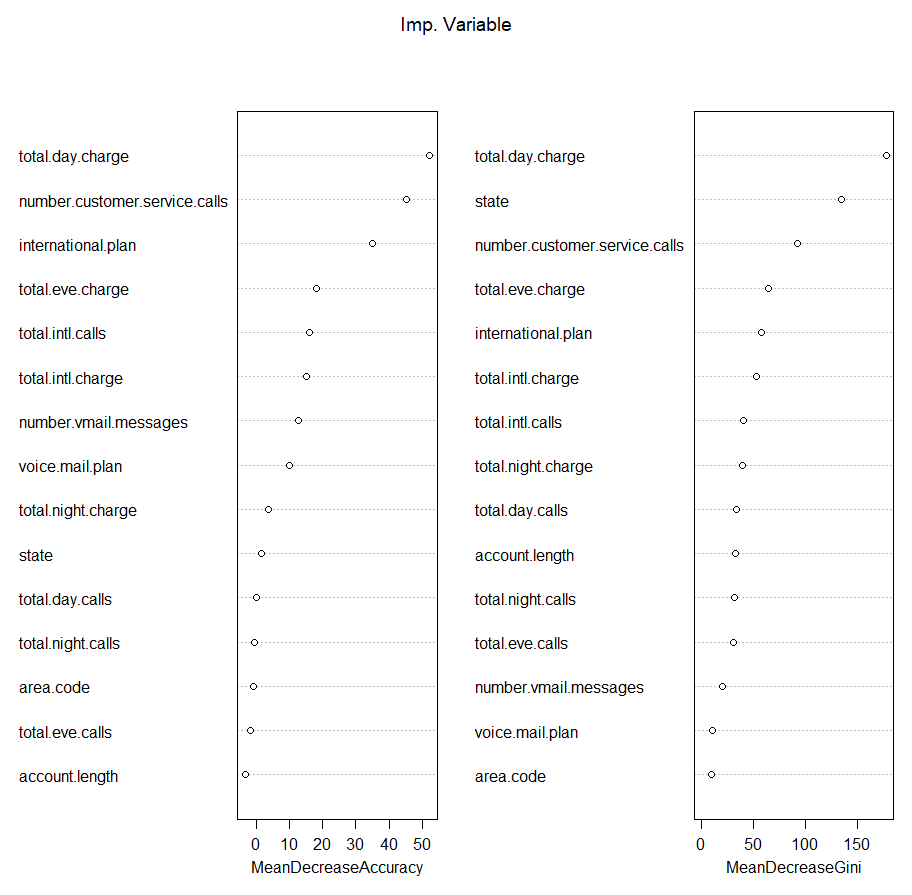
****

**Figure 2.9 Number for trees in used in Random Forest**

As here we can observe after n=200 trees are being used for model creation but after 200 the error rate is kind of constant so we will be taking n=200 for our Random Forest Model Development.

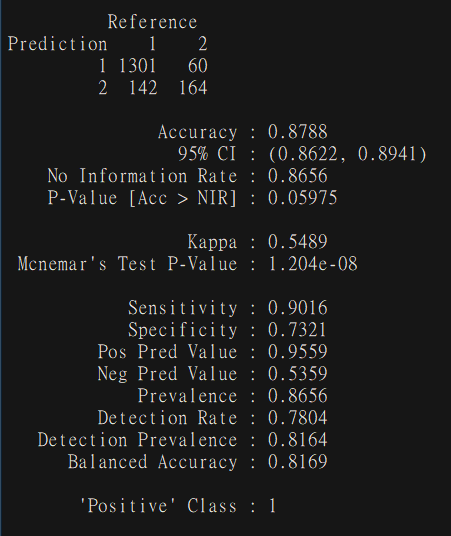


**Figure 2.10. Nodes and Number of tree in Random Forest model**

****

**Figure 2.11. Important Variable in Random Forest model**

The Accuracy obtained by the model is shown below,

****

**Figure 2.12. Random Forest Accuracy**

**Logistic Regression**

Logistic regression is another technique borrowed by machine learning from the field of statistics. It is the go-to method for binary classification problems (problems with two class values). The coefficients (Beta values b) of the logistic regression algorithm must be estimated from your training data. This is done using maximum-likelihood estimation. Logistic regression is a regression algorithm used to conduct when the dependent variable is binary where the dependent variable has more than two outcome categories will get analyzed in multinomial logistic regression, or, if the multiple categories are ordered, in ordinal logistic regression. Logistic regression also used to describe data and determine the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables. The summary of logistic model can be seen as following:

Call:

glm(formula = Churn ~ ., family = "binomial", data = train)

Deviance Residuals:

Min 1Q Median 3Q Max

-1.9135 -0.4977 -0.3120 -0.1659 3.0484

Coefficients:

Estimate Std. Error z value Pr(>|z|)

(Intercept) -9.64158 0.94641 -10.188 < 2e-16 \*\*\*

state2 0.33591 0.76417 0.440 0.660242

state3 0.90858 0.75353 1.206 0.227907

state4 0.12055 0.84475 0.143 0.886523

state5 1.82774 0.78219 2.337 0.019455 \*

state6 0.67607 0.76260 0.887 0.375329

state7 1.02180 0.72631 1.407 0.159477

state8 0.69517 0.80924 0.859 0.390319

state9 0.76200 0.74959 1.017 0.309366

state10 0.59574 0.76159 0.782 0.434081

state11 0.67805 0.77818 0.871 0.383574

state12 -0.21226 0.89560 -0.237 0.812657

state13 0.23602 0.90325 0.261 0.793857

state14 0.87469 0.74790 1.170 0.242197

state15 -0.20620 0.83322 -0.247 0.804542

state16 0.44382 0.75382 0.589 0.556026

state17 1.07190 0.73040 1.468 0.142229

state18 0.80540 0.76596 1.051 0.293036

state19 0.56522 0.83594 0.676 0.498945

state20 1.17571 0.74362 1.581 0.113865

state21 1.14453 0.71710 1.596 0.110481

state22 1.35342 0.72832 1.858 0.063131 .

state23 1.38801 0.71413 1.944 0.051940 .

state24 1.17067 0.71585 1.635 0.101974

state25 0.59902 0.77472 0.773 0.439402

state26 1.36003 0.72798 1.868 0.061731 .

state27 1.87028 0.71735 2.607 0.009128 \*\*

state28 0.60716 0.75459 0.805 0.421041

state29 0.15582 0.79713 0.195 0.845017

state30 0.32490 0.80534 0.403 0.686631

state31 1.19175 0.76847 1.551 0.120951

state32 1.59468 0.70979 2.247 0.024660 \*

state33 0.47528 0.78759 0.603 0.546203

state34 1.25400 0.72542 1.729 0.083869 .

state35 1.16716 0.72037 1.620 0.105184

state36 0.68686 0.74724 0.919 0.357996

state37 0.88256 0.75423 1.170 0.241942

state38 0.78009 0.73631 1.059 0.289392

state39 1.15983 0.77995 1.487 0.137001

state40 -0.10247 0.81983 -0.125 0.900530

state41 1.77941 0.73736 2.413 0.015813 \*

state42 0.83526 0.76194 1.096 0.272981

state43 0.28253 0.82136 0.344 0.730858

state44 1.65240 0.70834 2.333 0.019659 \*

state45 1.05006 0.74417 1.411 0.158228

state46 -0.43502 0.82288 -0.529 0.597044

state47 0.10104 0.77844 0.130 0.896728

state48 1.42380 0.72465 1.965 0.049437 \*

state49 0.28028 0.78093 0.359 0.719666

state50 0.58562 0.73346 0.798 0.424620

state51 0.30294 0.75489 0.401 0.688202

account.length 0.23625 0.34713 0.681 0.496135

area.code -0.06218 0.13775 -0.451 0.651682

international.plan2 2.18813 0.15328 14.275 < 2e-16 \*\*\*

voice.mail.plan2 -2.10715 0.59311 -3.553 0.000381 \*\*\*

number.vmail.messages 1.91107 0.94922 2.013 0.044082 \*

total.day.calls 0.66721 0.47156 1.415 0.157102

total.day.charge 4.59878 0.38909 11.819 < 2e-16 \*\*\*

total.eve.calls 0.16764 0.49097 0.341 0.732772

total.eve.charge 2.82487 0.43058 6.561 5.36e-11 \*\*\*

total.night.calls 0.02657 0.41536 0.064 0.948990

total.night.charge 1.46040 0.42800 3.412 0.000645 \*\*\*

total.intl.calls -1.79972 0.51370 -3.503 0.000459 \*\*\*

total.intl.charge 1.67077 0.42178 3.961 7.46e-05 \*\*\*

number.customer.service.calls 4.83059 0.36840 13.112 < 2e-16 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 2758.3 on 3332 degrees of freedom

Residual deviance: 2072.0 on 3268 degrees of freedom

AIC: 2202

Number of Fisher Scoring iterations: 6

Actual

Predicted 1 2

1 1374 158

2 69 66

Here we can see Model have the Accuracy of 86.3%

**KNN Implementation**

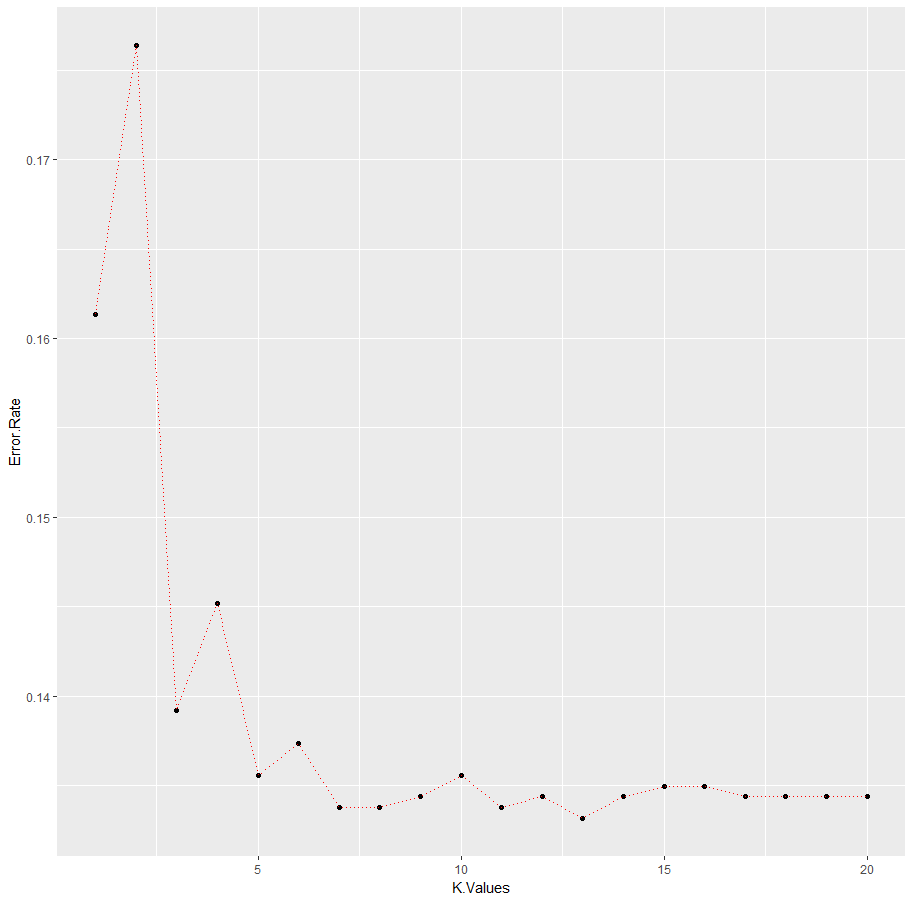
K-Nearest neighbors algorithm (k-NN) is a [non-parametric](https://en.wikipedia.org/wiki/Non-parametric_statistics) method used for [classification](https://en.wikipedia.org/wiki/Statistical_classification) and [regression](https://en.wikipedia.org/wiki/Regression_analysis). In both cases, the input consists of the k closest training examples in the [feature space](https://en.wikipedia.org/wiki/Feature_space). The output depends on whether k-NN is used for classification or regression:

* In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k-nearest neighbors (k is a positive [integer](https://en.wikipedia.org/wiki/Integer), typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.
* In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

k-NN is a type of [instance-based learning](https://en.wikipedia.org/wiki/Instance-based_learning), or [lazy learning](https://en.wikipedia.org/wiki/Lazy_learning), where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all [machine learning](https://en.wikipedia.org/wiki/Machine_learning) algorithms.

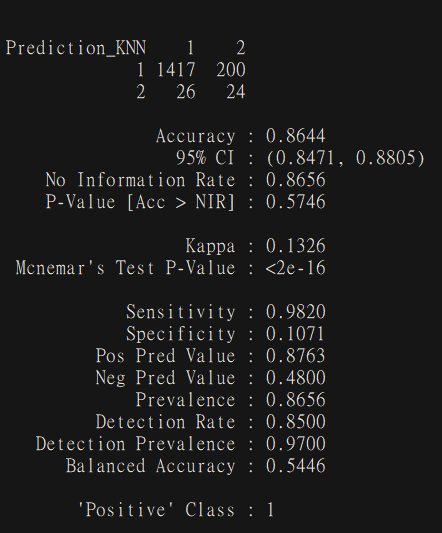
Both for classification and regression, a useful technique can be to assign weight to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of 1/d, where d is the distance to the neighbor.

For better K Value for our process we will calculate Error Rate for K = 1 to 20 and which ever K value turns out to be less we will take that as K value, and we can visualize the K and error rate for better understanding and for selection of K value.



**Figure 2.13. Error Rate Vs K Value**

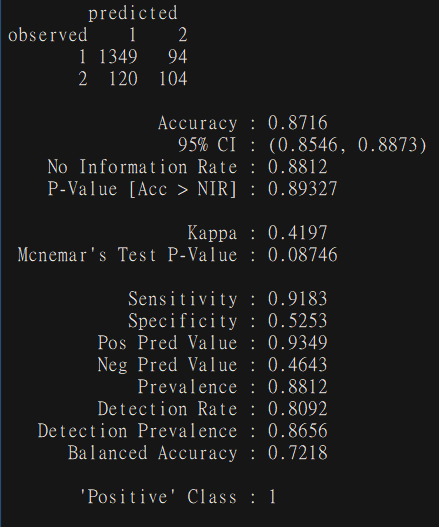
Here we can see K= 5 seems as better choice according to elbow method and have Error rate of 0.13 or less and after that we don’t see much of difference in error rate. So, we will be considering K=5 for Model development.



**Figure 2.14. KNN Accuracy**

**Naïve Bayes**

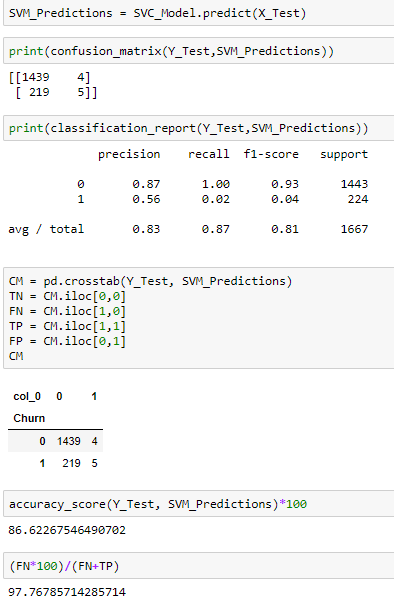
Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of [feature](https://en.wikipedia.org/wiki/Feature_vector) values, where the class labels are drawn from some finite set. It is not a single [algorithm](https://en.wikipedia.org/wiki/Algorithm) for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is [independent](https://en.wikipedia.org/wiki/Independence_(probability_theory)) of the value of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible [correlations](https://en.wikipedia.org/wiki/Correlation_and_dependence) between the color, roundness, and diameter features. For some types of probability models, naive Bayes classifiers can be trained very efficiently in a [supervised learning](https://en.wikipedia.org/wiki/Supervised_learning) setting. In many practical applications, parameter estimation for naive Bayes models uses the method of [maximum likelihood](https://en.wikipedia.org/wiki/Maximum_likelihood); in other words, one can work with the naive Bayes model without accepting [Bayesian probability](https://en.wikipedia.org/wiki/Bayesian_probability) or using any Bayesian methods. Despite their naive design and apparently oversimplified assumptions, naive Bayes classifiers have worked quite well in many complex real-world situations.



**Figure 2.14. Naïve Bayes Accuracy**

**SVM (Support Vector Machine)**

Support Vector Machine (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well.



**Figure 2.15. SVM Accuracy**

**2.3 Cross Validation**

Cross-validation is a technique to evaluate predictive models by partitioning the original sample into a training set to train the model, and a test set to evaluate it.   
In k-fold cross-validation, the original sample is randomly partitioned into k equal size subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k-1 subsamples are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k subsamples used exactly once as the validation data. The k results from the folds can then be averaged (or otherwise combined) to produce a single estimation.

The advantage of this method is that all observations are used for both training and validation, and each observation is used for validation exactly once.  
For classification problems, one typically uses stratified k-fold cross-validation, in which the folds are selected so that each fold contains roughly the same proportions of class labels.  
In repeated cross-validation, the cross-validation procedure is repeated n times, yielding n random partitions of the original sample.

The n results are again averaged (or otherwise combined) to produce a single estimation.   
Cross Validation will generate train-test splits given the number of folds and repeats, so that different users can evaluate their models with the same splits. Stratification is applied by default for classification problems (unless otherwise specified). The splits are given as part of the task description as an ARFF file with the row id, fold number, repeat number and the class (TRAIN or TEST). The uploaded predictions should be labeled with the fold and repeat number of the test instance, so that the results can be properly evaluated.

**Table 2.1 CROSS VALIDATION MEAN ACCURACY TABLE FOR FOLD = 10**

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Algorithm | CV Accuracy | Act. Accuracy |
| 1. | **DECISION TREE** | **95.4 %** | **88.9 %** |
| 2. | **RANDOM FOREST** | **94.1 %** | **87.8 %** |
| 3. | **LOGISTIC REG.** | **85.8 %** | **86.3 %** |
| 4. | **KNN Model** | **84.8 %** | **86.4 %** |
| 5. | **SVM Model** | **87.2 %** | **88.9 %** |
| 6. | **Naïve Bayes** | **87.3 %** | **87.1 %** |

Here we can observe that our model is working pretty well with more reliable accuracy even with cross validation, Most of the time accuracy tends to increase but it’s reliable as there is not a huge difference between Actual Accuracy of Model and Cross Validation Mean Accuracy.

**CHAPTER 3**

**CONCLUSION**

**3.1. Model Evaluation**

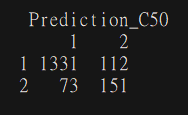
Model evaluation will be done on the basis of evaluation metrics or error matrices, it explains the performance of our model or any machine learning model. It determines an important aspects of the model and capability. Simply developing model is not the important part but to evaluate error rate and all metrics based on Confusion Matrix such as

* Accuracy [(TP+TN)/(TP+TN+FP+FN)]
* Sensitivity [TP/TP+FP]
* Specificity [TN/TN+FN]
* False Positive Rate [FP/FP+TN]
* False Negative Rate [FN/FN+TP]

Which will be based on Confusion Matrix Labels such as

* True Positive (TP)
* True Negative (TN)
* False Positive (FP)
* False Negative (FN)
* True Positive is the number of correct predictions that an instance is Yes.
* False Negative is the number of incorrect predictions that an instance is No.
* False Positive is the number of incorrect of predictions that an instance Yes.
* True Negative is the number of correct predictions that an instance is No.

**Decision Tree**

****

**Confusion Matrix**

Accuracy [(TP+TN)/(TP+TN+FP+FN)] = 88.9 %

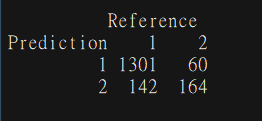
Sensitivity [TP/TP+FP] = 57.4 %

Specificity [TN/TN+FN] = 94.8 %

False Positive Rate [FP/FP+TN] = 7.76 %

False Negative Rate [FN/FN+TP] = 32.5 %

**Random Forest**

****

**Confusion Matrix**

Accuracy [(TP+TN)/(TP+TN+FP+FN)] = 87.8 %

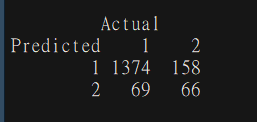
Sensitivity [TP/TP+FP] = 73.2 %

Specificity [TN/TN+FN] = 90.1 %

False Positive Rate [FP/FP+TN] = 4.4 %

False Negative Rate [FN/FN+TP] = 46.4 %

**Logistic Regression**

****

**Confusion Matrix**

Accuracy [(TP+TN)/(TP+TN+FP+FN)] = 86.3 %

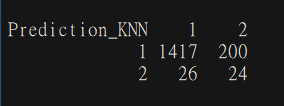
Sensitivity [TP/TP+FP] = 29.4 %

Specificity [TN/TN+FN] = 95.2 %

False Positive Rate [FP/FP+TN] = 10.3 %

False Negative Rate [FN/FN+TP] = 51.1 %

**K-Nearest Neighbors**

****

**Confusion Matrix**

Accuracy [(TP+TN)/(TP+TN+FP+FN)] = 86.4 %

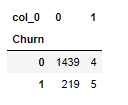
Sensitivity [TP/TP+FP] = 10.7 %

Specificity [TN/TN+FN] = 98.1 %

False Positive Rate [FP/FP+TN] = 12.3 %

False Negative Rate [FN/FN+TP] = 52 %

**Support Vector Machine**

****

**Confusion Matrix**

Accuracy [(TP+TN)/(TP+TN+FP+FN)] = 86.6 %

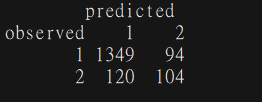
Sensitivity [TP/TP+FP] = 55.5 %

Specificity [TN/TN+FN] = 86.7 %

False Positive Rate [FP/FP+TN] = 0.2 %

False Negative Rate [FN/FN+TP] = 97.7 %

**Naïve Bayes**

****

**Confusion Matrix**

Accuracy [(TP+TN)/(TP+TN+FP+FN)] = 87.16 %

Sensitivity [TP/TP+FP] = 52.5 %

Specificity [TN/TN+FN] = 91.8 %

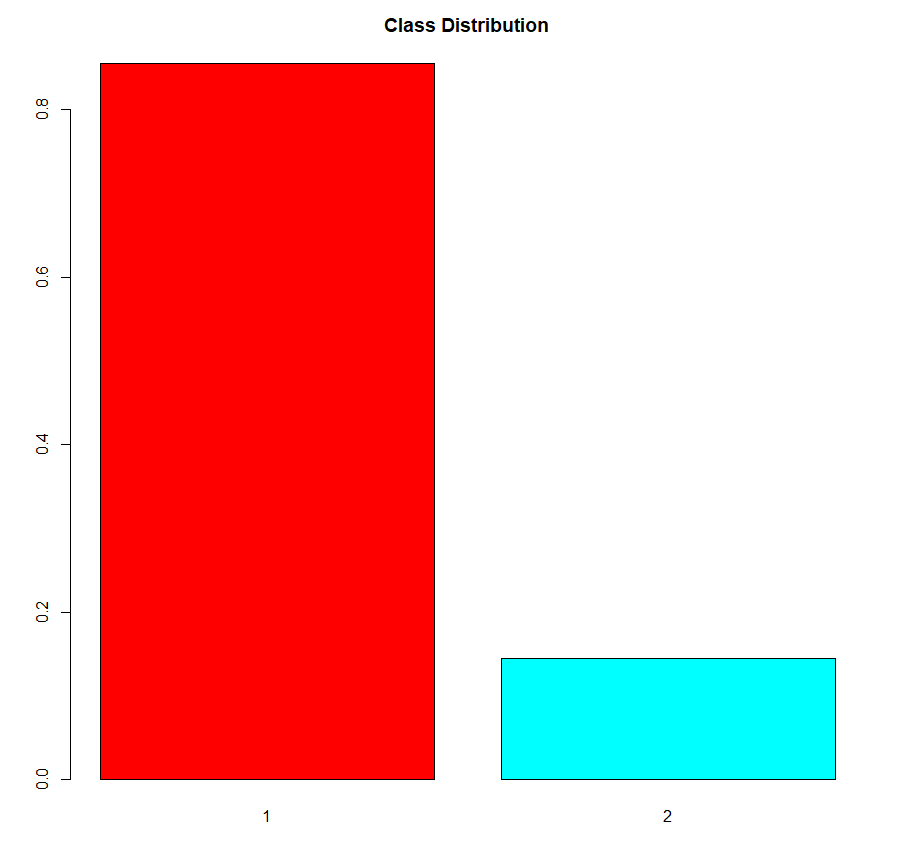
False Positive Rate [FP/FP+TN] = 6.5 %

False Negative Rate [FN/FN+TP] = 53.5 %

**3.2. Model Selection**

Now After evaluating error metrices on every model we came to know for our Dataset Decision perform well with Highest Accuracy of 88.9 % and False Negative Rate of 32.5 % and Second, we can consider Random forest as it has second height Accuracy of 87.8 % with False Negative Rate of 46.4 %, But after evaluating we found out that our problem have Class imbalance problem and we need to find the accuracy, Sensitivity, Specificity for our best model based on Accuracy, that is Decision Tree. We will consider Decision tree as our primary model for class imbalance problem and will see if we can increase our Sensitivity of 57.4 % to any more for Class “2” in our Dataset which Defines True as positive reaction of customer for churn.

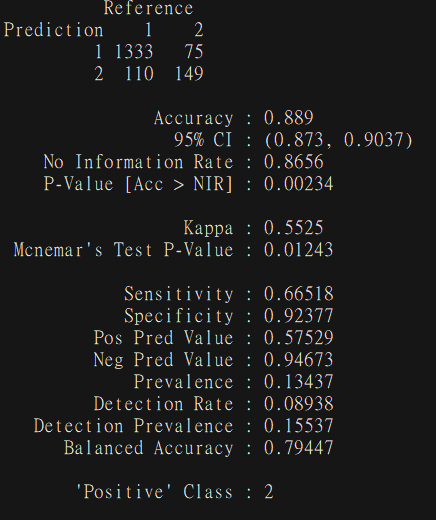
**3.3 Class Imbalance Problem**

****

**Figure 3.1. Class Distribution of Target Variable of Train Data**

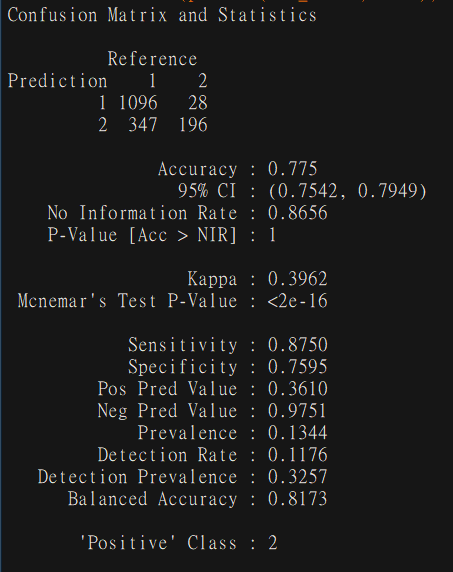
As we can see there is so much difference between 1 (False) & 2 (True) class in distribution. So, we are going to over fit and see the sensitivity and Accuracy for problem.

**Over Fitting**

****

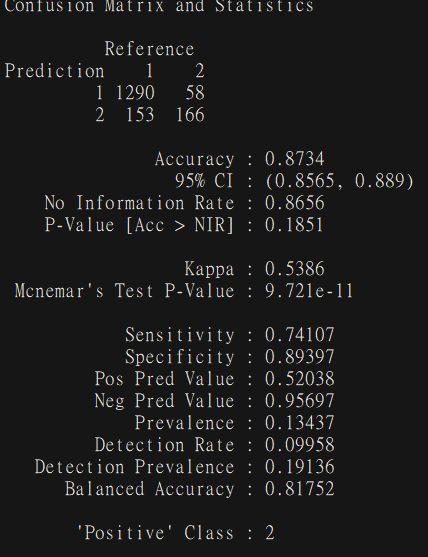
**Figure 3.2. Over Fitting class imbalance Accuracy in Decision Tree Model**

**Under Fitting**

****

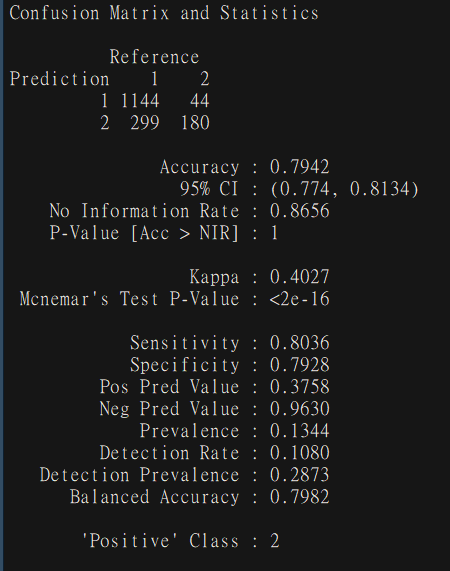
**Figure 3.3. Under Fitting class imbalance Accuracy in Decision Tree Model**

**Both Together**

****

**Figure 3.4. Both Under and Over Fitting class imbalance Accuracy in Decision Tree Model**

**Using ROSE (Random Over-Sampling Examples)**

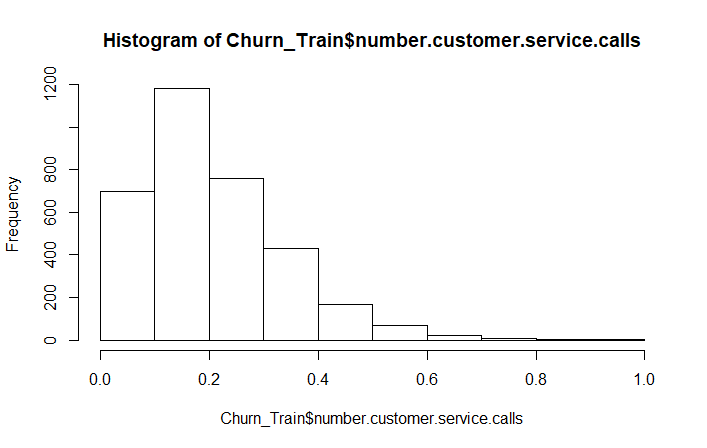
****

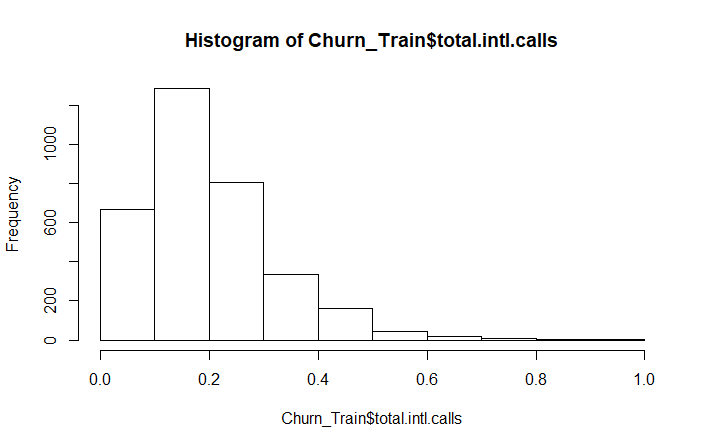
**Figure 3.5. Using ROSE class imbalance Accuracy in Decision Tree Model**

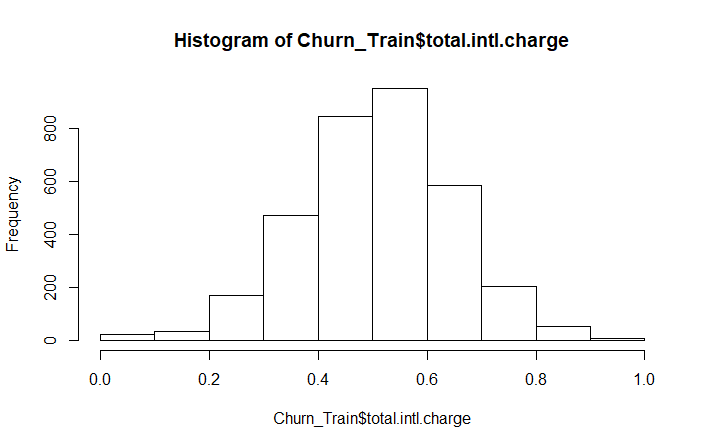
As we can observe if we are using both Under and Over fitting together on data which approximately diving Class equally and helping us to get 74.1 % Sensitivity without losing much accuracy i.e 87.3 % which was earlier 88.9 % and which is working both class distribution to gain more accurate model with more reliability.

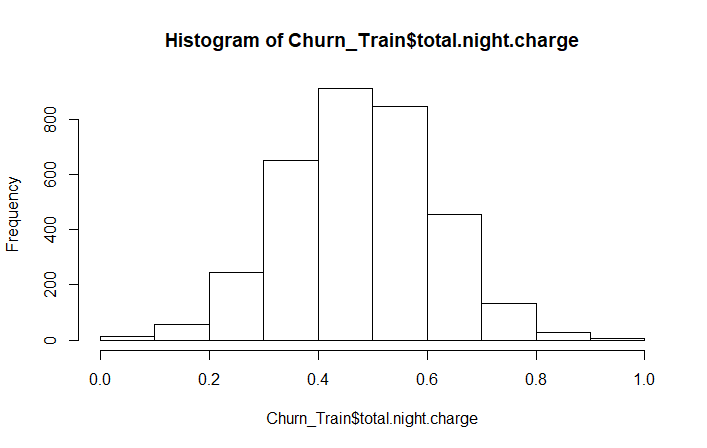
**Appendix A - Extra Figures**

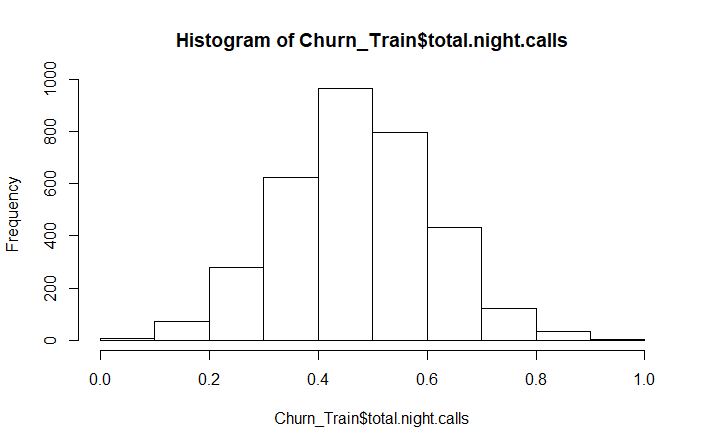
Normality check plots of various numerical variables:

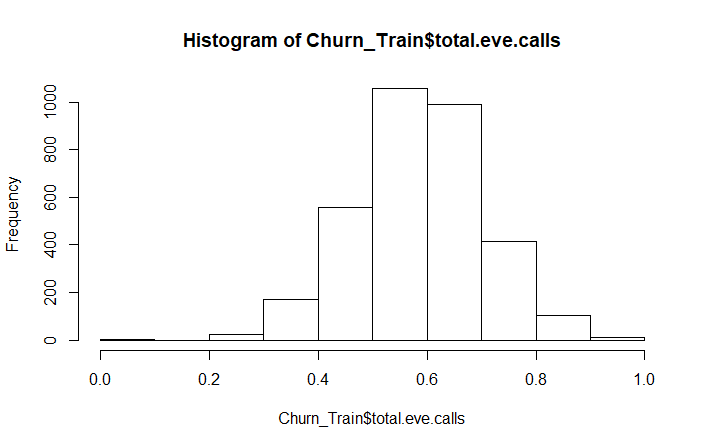
****

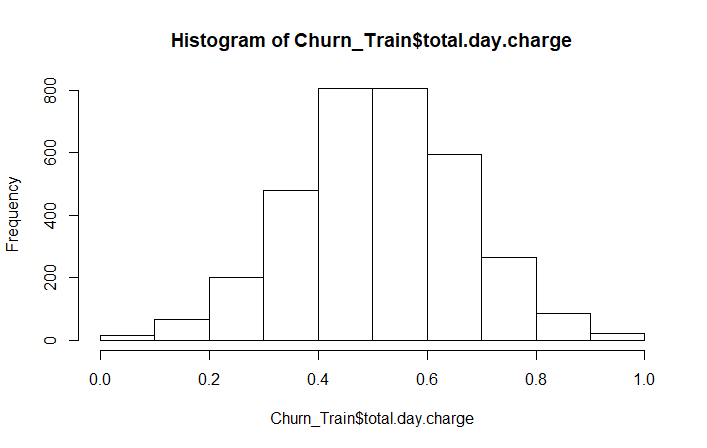
****

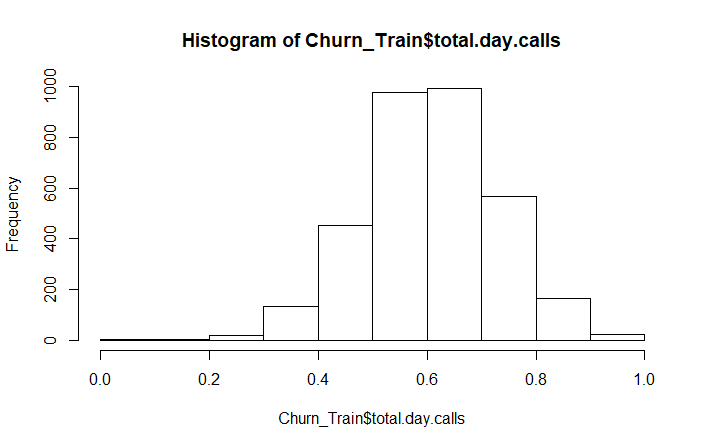
****

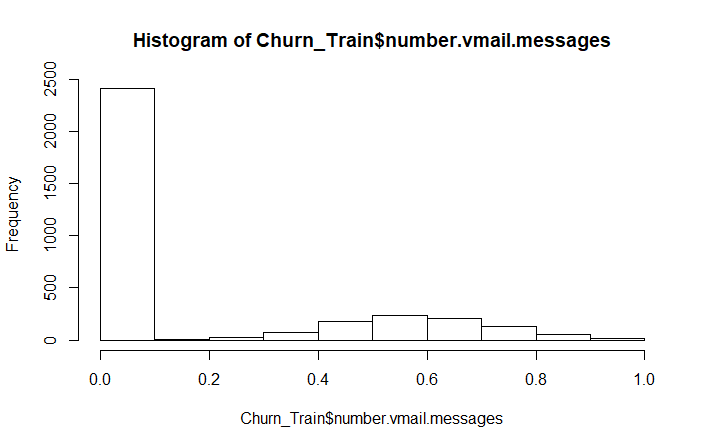
****

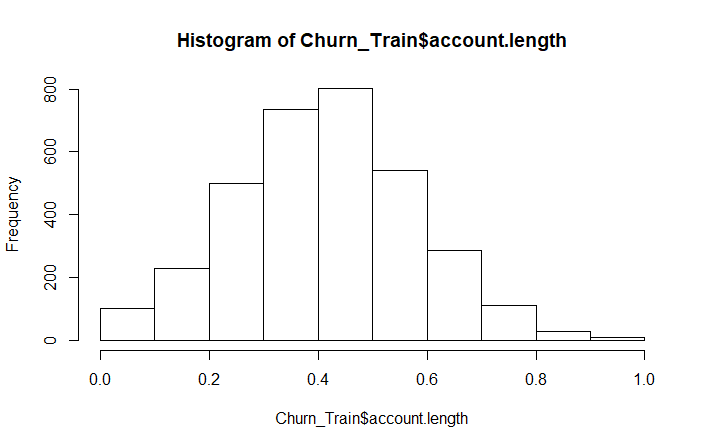
****

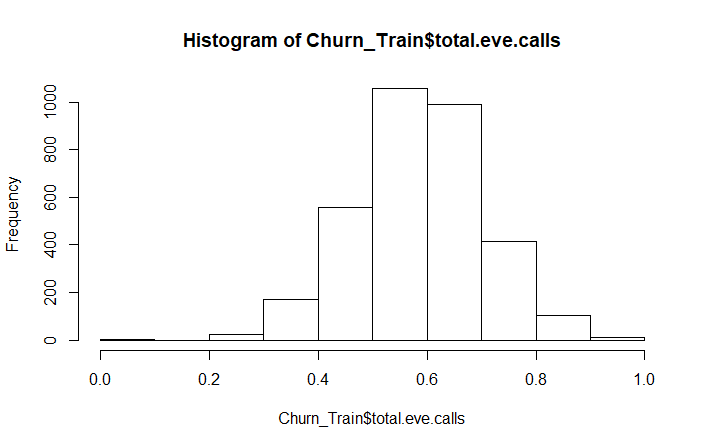
****

****

****

****

****

****

**Appendix B – Code**

**R Code**

# Loading Important Libraries for The Project

x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unbalanced", "C50", "dummies", "e1071", "Information",

"MASS", "rpart", "gbm", "ROSE", 'sampling', 'DataCombine', 'inTrees')

lapply(x, require, character.only = TRUE)

rm(x)

# Reading Data for Analysis

library('readxl')

Chunk\_Test <- read.csv('Churn\_Test.csv')

Chunk\_Train <- read.csv('Churn\_Train.csv')

str(Chunk\_Train)

# Missing Data Analysis in Both Test and Train data

missing <- data.frame(apply(Chunk\_Train,2,function(x){sum(is.na(x))}))

missing.test <- data.frame(apply(Chunk\_Test,2,function(x){sum(is.na(x))}))

# Convertion of Variable to Numeric

Chunk\_Train$phone.number <- as.numeric(Chunk\_Train$phone.number)

Chunk\_Test$phone.number <- as.numeric(Chunk\_Test$phone.number)

# Convertion of Factor And Categorical data into Factor for Both Test and Train data

Factor\_Name = c("state","international.plan","voice.mail.plan","Churn")

for(i in Factor\_Name){

if(class(Chunk\_Train[,i])== 'factor'){

Chunk\_Train[,i] = factor(Chunk\_Train[,i], labels = (1:length(levels(factor(Chunk\_Train[,i])))))

}

}

for(i in Factor\_Name){

if(class(Chunk\_Test[,i])== 'factor'){

Chunk\_Test[,i] = factor(Chunk\_Test[,i], labels = (1:length(levels(factor(Chunk\_Test[,i])))))

}

}

str(Chunk\_Train)

# Storing Numerical Variable in Factor\_Data for further analysis

Numeric\_Index = sapply(Chunk\_Train,is.numeric) #selecting only numeric

Numeric\_Data = Chunk\_Train[,Numeric\_Index]

Numerical = colnames(Numeric\_Data)

Numeric\_IndexTest = sapply(Chunk\_Test,is.numeric) #test data

Numeric\_TestData = Chunk\_Test[,Numeric\_IndexTest]

Numerical\_Test=colnames(Numeric\_TestData)

# Correlation Plot for Feature Selection

library('corrgram')

corrgram(Chunk\_Train[,Numeric\_Index], order = F,

upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")

corrgram(Chunk\_Test[,Numeric\_IndexTest], order = F,

upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot Test")

# Storing Factor Variable in Factor\_Data for further analysis

Factor\_Index = sapply(Chunk\_Train,is.factor)

Factor\_Data = Chunk\_Train[,Factor\_Index]

options(warn = -1)

# For loop to calculate Chi Sq. for Factor Variable in Dataset for Feature Selection

for (i in 1:4)

{

print(names(Factor\_Data)[i])

print(chisq.test(table(Factor\_Data$Churn,Factor\_Data[,i])))

}

# Dropping all Unneccessary data based on Correlation Plot and method

Churn\_Train = subset(Chunk\_Train, select = -c(total.day.minutes, total.eve.minutes, total.night.minutes, total.intl.minutes, phone.number))

Churn\_Test = subset(Chunk\_Test, select = -c(total.day.minutes, total.eve.minutes, total.night.minutes, total.intl.minutes, phone.number))

# Creating a Lisr containing all the Numerical or Continuous variable in the data set

Continuous\_Name = c("account.length","area.code","number.vmail.messages","total.day.calls","total.day.charge",

"total.eve.calls","total.eve.charge","total.night.calls","total.night.charge","total.intl.calls", "total.intl.charge",

"number.customer.service.calls")

# Normalization for Train Data

for(i in Continuous\_Name){

print(i)

Churn\_Train[,i] = (Churn\_Train[,i] - min(Churn\_Train[,i]))/

(max(Churn\_Train[,i] - min(Churn\_Train[,i])))

}

# Normalization for Test Data

for(i in Continuous\_Name){

print(i)

Churn\_Test[,i] = (Churn\_Test[,i] - min(Churn\_Test[,i]))/

(max(Churn\_Test[,i] - min(Churn\_Test[,i])))

#-----------------------------------------------------------------------------#

#### Histogram Plot For Continuous Data After Normalization Train Data ####

#-----------------------------------------------------------------------------#

qqnorm(Churn\_Train$account.length)

hist(Churn\_Train$account.length)

qqnorm(Churn\_Train$number.vmail.messages)

hist(Churn\_Train$number.vmail.messages)

qqnorm(Churn\_Train$total.day.calls)

hist(Churn\_Train$total.day.calls)

qqnorm(Churn\_Train$total.day.charge)

hist(Churn\_Train$total.day.charge)

qqnorm(Churn\_Train$total.eve.calls)

hist(Churn\_Train$total.eve.calls)

qqnorm(Churn\_Train$total.eve.charge)

hist(Churn\_Train$total.eve.charge)

qqnorm(Churn\_Train$total.night.calls)

hist(Churn\_Train$total.night.calls)

qqnorm(Churn\_Train$total.night.charge)

hist(Churn\_Train$total.night.charge)

qqnorm(Churn\_Train$total.intl.charge)

hist(Churn\_Train$total.intl.charge)

qqnorm(Churn\_Train$total.intl.calls)

hist(Churn\_Train$total.intl.calls)

qqnorm(Churn\_Train$number.customer.service.calls)

hist(Churn\_Train$number.customer.service.calls)

#-----------------------------------------------------------------------------#

#### Histogram Plot For Continuous Data After Normalization Test Data ####

#-----------------------------------------------------------------------------#

qqnorm(Churn\_Test$account.length)

hist(Churn\_Test$account.length)

qqnorm(Churn\_Test$number.vmail.messages)

hist(Churn\_Test$number.vmail.messages)

qqnorm(Churn\_Test$total.day.calls)

hist(Churn\_Test$total.day.calls)

qqnorm(Churn\_Test$total.day.charge)

hist(Churn\_Test$total.day.charge)

qqnorm(Churn\_Test$total.eve.calls)

hist(Churn\_Test$total.eve.calls)

qqnorm(Churn\_Test$total.eve.charge)

hist(Churn\_Test$total.eve.charge)

qqnorm(Churn\_Test$total.night.calls)

hist(Churn\_Test$total.night.calls)

qqnorm(Churn\_Test$total.night.charge)

hist(Churn\_Test$total.night.charge)

qqnorm(Churn\_Test$total.intl.charge)

hist(Churn\_Test$total.intl.charge)

qqnorm(Churn\_Test$total.intl.calls)

hist(Churn\_Test$total.intl.calls)

qqnorm(Churn\_Test$number.customer.service.calls)

hist(Churn\_Test$number.customer.service.calls)

rmExcept(c("Churn\_Train","Churn\_Test"))

# Copying Churn\_Train to train and Churn\_Test to test For better understanding of Data for ML Alg.

train = Churn\_Train

test = Churn\_Test

# Decision Tree Model On C5.0

C50\_Dtree = C5.0(Churn ~., train, trials = 50, rules = TRUE)

summary(C50\_Dtree)

# Writing all summary and Rules in Txt for better understanding of Model

write(capture.output(summary(C50\_Dtree)), "C50\_Dtree.txt")

Prediction\_C50 = predict(C50\_Dtree, test[,-16], type = "class")

CM\_C50 = table(test$Churn, Prediction\_C50)

confusionMatrix(CM\_C50, positive = '2')

Accuracy\_DT = (CM\_C50[1,1]+CM\_C50[2,2])/(CM\_C50[1,1]+CM\_C50[2,2]+CM\_C50[1,2]+CM\_C50[2,1])

library('caret')

Folds = createFolds(train$Churn, k = 10)

CV\_DT = lapply(Folds, function(x){

train\_fold = train[-x, ]

test\_fold = train[x, ]

C50\_Dtree = C5.0(Churn ~., train\_fold, trials = 50, rules = TRUE)

Prediction\_C50 = predict(C50\_Dtree, test\_fold[,-16], type = "class")

CM\_C50 = table(test\_fold$Churn, Prediction\_C50)

confusionMatrix(CM\_C50, positive = '2')

accuracy\_dt = (CM\_C50[1,1]+CM\_C50[2,2])/(CM\_C50[1,1]+CM\_C50[2,2]+CM\_C50[1,2]+CM\_C50[2,1])

return(accuracy\_dt)

})

CV\_Accuracy\_DT = mean(as.numeric(CV\_DT))

#-------- BarPlot to visualize Class Imbalance Problem -------#

barplot(prop.table(table(train$Churn)),

col = rainbow(2),

main = "Class Distribution")

#-------- Importing ROSE (Random Over-Sampling Examples) Library for Class Imbalance -------#

library(ROSE)

#-------- Over Fitting of Class "2" for Better Sesitivity -------#

over <- ovun.sample(Churn~., data = train, method = "over", N = 5700)$data

table(over$Churn)

#-------- Model Creation and Confusion Matrix-------#

C50\_Over <- C5.0(Churn ~., data = over, trials = 50, rules = TRUE)

confusionMatrix(predict(C50\_Over, test), test$Churn, positive = '2')

#-------- Under Fitting of Class "1" for Better Sesitivity -------#

under <- ovun.sample(Churn~., data = train, method = "under", N = 966)$data

table(under$Churn)

#-------- Model Creation and Confusion Matrix-------#

C50\_Under <- C5.0(Churn ~., data = under, trials = 50, rules = TRUE)

confusionMatrix(predict(C50\_Under, test), test$Churn, positive = '2')

#-------- Applying Both Under and Over Fitting for Better Sesitivity -------#

both <- ovun.sample(Churn~., data = train, method = "both",

N = 3333)$data

table(both$Churn)

#-------- Model Creation and Confusion Matrix-------#

C50\_Both <- C5.0(Churn ~., data = both, trials = 50, rules = TRUE)

confusionMatrix(predict(C50\_Both, test), test$Churn, positive = '2')

#-------- Applying Both Under and Over Fitting for Better Sesitivity Using ROSE -------#

rose <- ROSE(Churn~., data = train, N = 3333)$data

table(rose$Churn)

#-------- Model Creation and Confusion Matrix-------#

C50\_Rose <- C5.0(Churn ~., data = rose, trials = 50, rules = TRUE)

confusionMatrix(predict(C50\_Rose, test), test$Churn, positive = '2')

RF\_Model <- randomForest(Churn ~.,data = train,

importance = TRUE,

proximity = T,

ntree = 500)

print(RF\_Model)

attributes(RF\_Model)

plot(RF\_Model)

RF\_Model <- randomForest(Churn ~.,data = train,

importance = TRUE,

proximity = T,

ntree = 200)

# Histogram For Number Of Nodes For the Trees

hist(treesize(RF\_Model),

main = "No. of Nodes for the Trees",

col = "blue")

# Plottting Important Variable

varImpPlot(RF\_Model,

sort = T,

main = 'Imp. Variable')

# Important Variable in Random Forest Based on MDAccuracy and Gini

importance(RF\_Model)

# Most Varible Used While Creating Model

varUsed(RF\_Model)

getTree(RF\_Model,1,labelVar = T)

Prediction\_RF = predict(RF\_Model, test[,-16])

confusionMatrix(Prediction\_RF,test$Churn, positive = '2')

CV\_RF = lapply(Folds, function(x){

train\_fold\_RF = train[-x, ]

test\_fold\_RF = train[x, ]

RF\_Model <- randomForest(Churn ~.,data = train\_fold\_RF,

importance = TRUE,

proximity = T,

ntree = 200)

Prediction\_RF = predict(RF\_Model, test\_fold\_RF[,-16])

CM\_RF = table(test\_fold\_RF$Churn, Prediction\_RF)

confusionMatrix(CM\_RF, positive = '2')

accuracy\_rf = (CM\_RF[1,1]+CM\_RF[2,2])/(CM\_RF[1,1]+CM\_RF[2,2]+CM\_RF[1,2]+CM\_RF[2,1])

return(accuracy\_rf)

})

CV\_Accuracy\_RF = mean(as.numeric(CV\_RF))

treeList <- RF2List(RF\_Model)

Ext\_Rule = extractRules(treeList, train[,-16])

Ext\_Rule[1:2,]

Readable = presentRules(Ext\_Rule, colnames(train))

Readable[1:2,]

Rule\_Metric = getRuleMetric(Ext\_Rule, train[,-16], train$Churn)

Rule\_Metric[1:2,]

# Logistic Regression Model

Log\_Model <- glm(Churn ~.,data = train, family = 'binomial')

summary(Log\_Model)

#Prediction Based on Test Data

Predict\_Log <- predict(Log\_Model,test,type = 'response')

pred\_MissTest <- ifelse(Predict\_Log > 0.5,2,1)

# Confusion Matrix For Logistic Regression

CM\_LR <- table(Predicted = pred\_MissTest, Actual = test$Churn)

CM\_LR

1-sum(diag(CM\_LR))/sum(CM\_LR)

confusionMatrix(CM\_LR, positive = '2')

# Error Rate

with(Log\_Model,pchisq(null.deviance-deviance,df.null-df.residual,lower.tail = F))

Accuracy\_LR = (CM\_LR[1,1]+CM\_LR[2,2])/(CM\_LR[1,1]+CM\_LR[2,2]+CM\_LR[1,2]+CM\_LR[2,1])

CV\_LR = lapply(Folds, function(x){

train\_fold\_LR = train[-x, ]

test\_fold\_LR = train[x, ]

Log\_Model <- glm(Churn ~., data = train\_fold\_LR, family = 'binomial')

Predict\_Log <- predict(Log\_Model, test\_fold\_LR,type = 'response')

pred\_MissTest <- ifelse(Predict\_Log > 0.5,2,1)

CM\_LR <- table(Predicted = pred\_MissTest, Actual = test\_fold\_LR$Churn)

confusionMatrix(CM\_LR,positive = '2')

accuracy\_lr = (CM\_LR[1,1]+CM\_LR[2,2])/(CM\_LR[1,1]+CM\_LR[2,2]+CM\_LR[1,2]+CM\_LR[2,1])

return(accuracy\_lr)

})

CV\_Accuracy\_LR = mean(as.numeric(CV\_LR))

# CV LOG\_REG ACCURACY: 85.8%

library(class)

#Creating A List Of NULL For Prediction and Error Rate

Prediction\_KNN = NULL

Error.Rate = NULL

# For Loop to find Error Rate Based On K Between 1 to 20

for(i in 1:20){

Prediction\_KNN = knn(train[, 1:15], test[, 1:15], train$Churn, k=i)

Error.Rate[i] = mean(test$Churn != Prediction\_KNN)

}

print(Error.Rate)

K.Values <- 1:20

Error.DF <- data.frame(Error.Rate,K.Values)

# Plotting Of K Value And Error Rate For Best Suited K Value

ggplot(Error.DF ,aes(x=K.Values, y=Error.Rate)) + geom\_point()+ geom\_line(lty="dotted",color='red')\

#Predioction On Test Data

Prediction\_KNN = knn(train[, 1:15], test[, 1:15], train$Churn, k = 5)

head(Prediction\_KNN)

# Evaluation model for trained data and analysis of misclassification error rate.

mean(test$Churn != Prediction\_KNN)

#Confusion matrix

CM\_KNN = table(Prediction\_KNN , test$Churn)

confusionMatrix(CM\_KNN, positive = '2')

Accuracy\_KNN = (CM\_KNN[1,1]+CM\_KNN[2,2])/(CM\_KNN[1,1]+CM\_KNN[2,2]+CM\_KNN[1,2]+CM\_KNN[2,1])

CV\_KNN = lapply(Folds, function(x){

train\_fold\_KNN = train[-x, ]

test\_fold\_KNN = train[x, ]

Prediction\_KNN = knn(train\_fold\_KNN[, 1:15], test\_fold\_KNN[, 1:15], train\_fold\_KNN$Churn, k = 3)

mean(test\_fold\_KNN$Churn != Prediction\_KNN)

CM\_KNN = table(Prediction\_KNN , test\_fold\_KNN$Churn)

confusionMatrix(CM\_KNN,positive = '2')

accuracy\_knn = (CM\_KNN[1,1]+CM\_KNN[2,2])/(CM\_KNN[1,1]+CM\_KNN[2,2]+CM\_KNN[1,2]+CM\_KNN[2,1])

return(accuracy\_knn)

})

CV\_Accuracy\_KNN = mean(as.numeric(CV\_KNN))

#CV\_Accuracy\_KNN: 84.8 %

#SVM Model Creation

SVM\_Model <- svm(Churn ~ ., data=train)

summary(SVM\_Model)

Prediction\_SVM <- predict(SVM\_Model,test[,-16])SVM\_Tab <- table(Prediction\_SVM,test$Churn)

confusionMatrix(SVM\_Tab, positive = '2')

Accuracy\_SVM = (SVM\_Tab[1,1]+SVM\_Tab[2,2])/(SVM\_Tab[1,1]+SVM\_Tab[2,2]+SVM\_Tab[1,2]+SVM\_Tab[2,1])

CV\_SVM = lapply(Folds, function(x){

train\_fold\_SVM = train[-x, ]

test\_fold\_SVM = train[x, ]

SVM\_Model <- svm(Churn ~ ., data=train\_fold\_SVM)

Prediction\_SVM <- predict(SVM\_Model,test\_fold\_SVM[,-16])

SVM\_Tab <- table(Prediction\_SVM,test\_fold\_SVM$Churn)

confusionMatrix(SVM\_Tab,positive = '2')

accuracy\_svm = (SVM\_Tab[1,1]+SVM\_Tab[2,2])/(SVM\_Tab[1,1]+SVM\_Tab[2,2]+SVM\_Tab[1,2]+SVM\_Tab[2,1])

return(accuracy\_svm)

})

CV\_Accuracy\_SVM = mean(as.numeric(CV\_SVM))

# CV\_Accuracy\_SVM: 87.2%

library(e1071)

# Naive Bayes Model Accuracy

NB\_Model = naiveBayes(Churn ~ ., data = train)

# Prediction On Test Data

NB\_Prediction = predict(NB\_Model, test[,1:15], type = 'class')

# Confusion Matrix For Naive Bayes

CM\_NB = table(observed = test[,16], predicted = NB\_Prediction)

conusionMatrix(CM\_NB, positive = '2')

Accuracy\_NB = (CM\_NB[1,1]+CM\_NB[2,2])/(CM\_NB[1,1]+CM\_NB[2,2]+CM\_NB[1,2]+CM\_NB[2,1])

CV\_NB = lapply(Folds, function(x){

train\_fold\_NB = train[-x, ]

test\_fold\_NB = train[x, ]

NB\_Model = naiveBayes(Churn ~ ., data = train\_fold\_NB)

NB\_Prediction = predict(NB\_Model, test\_fold\_NB[,1:15], type = 'class')

CM\_NB = table(observed = test\_fold\_NB[,16], predicted = NB\_Prediction)

confusionMatrix(CM\_NB,positive = '2')

accuracy\_nb = (CM\_NB[1,1]+CM\_NB[2,2])/(CM\_NB[1,1]+CM\_NB[2,2]+CM\_NB[1,2]+CM\_NB[2,1])

return(accuracy\_nb)

})

CV\_Accuracy\_NB = mean(as.numeric(CV\_NB))

# CV\_Accuracy\_NB: 87.3 %

**Python Code**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

Churn\_Train = pd.read\_csv("Churn\_Train.csv")

Churn\_Test = pd.read\_csv("Churn\_Test.csv")

plt.figure(figsize=(6,4))

sns.heatmap(Churn\_Train.isnull(),yticklabels=False,cbar=False,cmap='viridis')

plt.figure(figsize=(6,4))

sns.heatmap(Churn\_Test.isnull(),yticklabels=False,cbar=False,cmap='viridis')

Y = Churn\_Train["Churn"].value\_counts()

sns.barplot(Y.index, Y.values, palette="rainbow")

Churn\_Train.groupby(["state", "Churn"]).size().unstack().plot(kind='bar', stacked=False, figsize=(30,15),cmap = 'rainbow')

Churn\_Train.groupby(["area code", "Churn"]).size().unstack().plot(kind='bar', stacked=False, figsize=(5,5),cmap = 'rainbow')

Churn\_Train.groupby(["international plan", "Churn"]).size().unstack().plot(kind='bar', stacked=False, figsize=(5,5))

Churn\_Train.groupby(["voice mail plan", "Churn"]).size().unstack().plot(kind='bar', stacked=False, figsize=(5,5))

for i in range(0, Churn\_Train.shape[1]):

if(Churn\_Train.iloc[:,i].dtypes == 'object'):

Churn\_Train.iloc[:,i] = pd.Categorical(Churn\_Train.iloc[:,i])

Churn\_Train.iloc[:,i] = Churn\_Train.iloc[:,i].cat.codes

for i in range(0, Churn\_Test.shape[1]):

if(Churn\_Test.iloc[:,i].dtypes == 'object'):

Churn\_Test.iloc[:,i] = pd.Categorical(Churn\_Test.iloc[:,i])

Churn\_Test.iloc[:,i] = Churn\_Test.iloc[:,i].cat.codes

Y\_Train = Churn\_Train.Churn

Y\_Test = Churn\_Test.Churn

Combine = Churn\_Train.append(Churn\_Test)

print(Combine.shape, Churn\_Train.shape, Churn\_Test.shape)

Numerical = ["account length","area code","number vmail messages","total day minutes","total day calls","total day charge",

"total eve minutes","total eve calls","total eve charge","total night minutes","total night calls",

"total night charge","total intl minutes","total intl calls", "total intl charge",

"number customer service calls"]

Df\_Corr = Combine.loc[:,Numerical]

Corr = Df\_Corr.corr()

plt.figure(figsize=(14, 12))

sns.heatmap(Corr, mask=np.zeros\_like(Corr,dtype=np.bool),cmap = 'rainbow',

square = True, annot = True)

Categorical = ["state","phone number","international plan","voice mail plan"]

from scipy.stats import chi2\_contingency

for i in Categorical:

print(i)

chi2, p, dof, ex = chi2\_contingency(pd.crosstab(Combine['Churn'],Combine[i]))

print(p)

Combine = Combine.drop(["total day minutes", "total eve minutes", "total night minutes", "total intl minutes",

"phone number","Churn"], axis = 1)

Numerical = ["account length","area code","number vmail messages","total day calls","total day charge",

"total eve calls","total eve charge","total night calls","total night charge","total intl calls",

"total intl charge", "number customer service calls"]

for i in Numerical:

print(i)

Combine[i] = (Combine[i]-min(Combine[i]))/(max(Combine[i])-min(Combine[i]))

from sklearn import tree

from sklearn.metrics import accuracy\_score

from sklearn.cross\_validation import train\_test\_split

X\_Train = Combine[:3333]

X\_Test = Combine[3333:]

X\_Train.shape

C50\_Model = tree.DecisionTreeClassifier(criterion = 'entropy')

C50\_Model.fit(X\_Train, Y\_Train)

from sklearn.metrics import confusion\_matrix,classification\_report

print("Confusion Matrix:")

print(confusion\_matrix(Y\_Test, C50\_Prediction))

print("\n")

print("Classification Report:")

print(classification\_report(Y\_Test, C50\_Prediction))

CM = pd.crosstab(Y\_Test, C50\_Prediction)

TN = CM.iloc[0,0]

FN = CM.iloc[1,0]

TP = CM.iloc[1,1]

FP = CM.iloc[0,1]

accuracy\_score(Y\_Test, C50\_Prediction)\*100

(FN\*100)/(FN+TP)

from sklearn.model\_selection import cross\_val\_score

CV\_Accuracy\_DT = cross\_val\_score(estimator= C50\_Model, X= X\_Train, y=Y\_Train, cv=10)

from sklearn.ensemble import RandomForestClassifier

RF\_Model = RandomForestClassifier()

RF\_Model.fit(X\_Train,Y\_Train)

RF\_Prediction = RF\_Model.predict(X\_Test)

print("Confusion Matrix:")

print(confusion\_matrix(Y\_Test, RF\_Prediction))

print("\n")

print("Classification Report:")

print(classification\_report(Y\_Test, RF\_Prediction))

CM = pd.crosstab(Y\_Test, RF\_Prediction)

TN = CM.iloc[0,0]

FN = CM.iloc[1,0]

TP = CM.iloc[1,1]

FP = CM.iloc[0,1]

CM

accuracy\_score(Y\_Test, RF\_Prediction)\*100

CV\_Accuracy\_RF.mean()

plt.figure(figsize=(10,6))

plt.plot(range(1,40),Error\_Rate,color='blue', linestyle='dashed', marker='o',

markerfacecolor='red', markersize=10)

plt.title('Error Rate vs. K Value')

plt.xlabel('K')

plt.ylabel('Error Rate')

# NOW WITH K=5

KNN = KNeighborsClassifier(n\_neighbors=5)

KNN.fit(X\_Train,Y\_Train)

Pred\_KNN = KNN.predict(X\_Test)

print('WITH K=5')

print('\n')

print(confusion\_matrix(Y\_Test,Pred\_KNN))

print('\n')

print(classification\_report(Y\_Test, Pred\_KNN))

accuracy\_score(Y\_Test, Pred\_KNN)\*100

CM = pd.crosstab(Y\_Test, Pred\_KNN)

TN = CM.iloc[0,0]

FN = CM.iloc[1,0]

TP = CM.iloc[1,1]

FP = CM.iloc[0,1]

CM

(FN\*100)/(FN+TP)

CV\_Accuracy\_KNN = cross\_val\_score(KNN, X=X\_Train, y=Y\_Train, cv=10)

CV\_Accuracy\_KNN.mean()