Churn Reduction Project

Report By Neha

January 18, 2019

1.Introduction

1.1 Project Description

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. This problem statement is targeted at enabling churn reduction using analytics concepts.

1.2 Problem statement

The objective of this Case is to predict customer behaviour. We are providing you a public dataset that has customer usage pattern and if the customer has moved or not. We expect you to develop an algorithm to predict the churn score based on usage pattern.
Target Variable : Churn: if the customer has moved (1=yes; 0 = no)

1.3 Data

Our task is to build classification models which will classify whether customer will move out or not depending on various factors given in the data. Given below is set of predictor variables given to classify the customer churn and the data for churn Reduction Classi cation.

state	account length	area code	phone number	international plan
HI	101	510	354-8815	no
MT	137	510	381-7211	no
OH	103	408	411-9481	no
NM	99	415	418-9100	no
SC	108	415	413-3643	no
IA	117	415	375-6180	no
ND	63	415	348-8073	no

Table 1.2: Churn Reduction sample Data (Columns 1-5)

voicemail plan	number vmail messages	total day minutes	total day calls	total day charge
no	0	70.9	123	12.05
no	0	223.6	86	38.01
yes	29	294.7	95	50.1
no	0	216.8	123	36.86
no	0	197.4	78	33.56
no	0	226.5	85	38.51
yes	32	218.9	124	37.21

Table 1.3: Churn Reduction sample Data (Columns 6-10)

total eve minutes	total eve calls	total eve charge	total night minutes	total night calls	total night charge
211.9	73	18.01	236	73	10.62
244.8	139	20.81	94.2	81	4.24
237.3	105	20.17	300.3	127	13.51
126.4	88	10.74	220.6	82	9.93
124	101	10.54	204.5	107	9.2
141.6	68	12.04	223	90	10.04
214.3	125	18.22	260.3	120	11.71

Table 1.4: Churn Reduction sample Data (Columns 11-16)

total intl minutes	total intl calls	total intl charge number	customer service calls	Churn
10.6	3	2.86	3	False.
9.5	7	2.57	0	False.
13.7	6	3.7	1	False.
15.7	2	4.24	1	False.
7.7	4	2.08	2	False.
6.9	5	1.86	1	False.
12.9	3	3.48	1	False.

Table 1.5: Churn Reduction sample Data (Columns 17-21)

2. Methodology

2.1 Pre Processing

Any model requires that we look at the data before we start modeling. However, in data mining terms looking at data refers to so much more than just looking. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as Exploratory Data Analysis. During this stage the data is explored and cleaned so as to make it fit for modelling. The data is visualized using different graphs to gain insight about it. Exploratory data analysis begins by exploring the class or data type of the different predictor variables.

The data is searched for presence of any missing values that can be either ignored (if more than 30% of data is missing) or imputed using different methods like mean, median, KNN (for numeric data) or mode (for categorical data). The variables are visualized to analyse their distribution (e.g. histograms can be used to visualize the distribution of variables). Outliers from the data are removed as they are inconsistent with the rest of the data. However, in some cases outliers provide interesting insights in cases like fraud detection, cancer detection etc and so sometimes they are not removed. Further variables are selected that contribute in future selection. Predictors that carry repetitive information are removed. Feature engineering may also be performed to generate new variables that will have a relation with the target variable. The following subsections will describe the pre-processing steps followed.

2.1.1 Missing value Analysis

The dataset obtained after combining the train and test data is checked for presence of missing values, however, no predictor showed presence of missing value.

2.1.2 Outlier Analysis

We can clearly observe from these probability distributions that most of the variables are skewed, for example, number customer service calls, number vmail messages, total day calls, total intl calls. The skew in these distributions

can be most likely explained by the presence of outliers and extreme values in the data.

One of the other steps of preprocessing apart from checking for normality is the presence of outliers. In this case we use a classic approach of outlier imputation, KNN imputation method. We visualize the outliers using boxplots. In figure 2.3 we have plotted the boxplots of the 14 continuous predictor variables with respect to each churn value. A lot of useful inferences can be made from these plots. As we can see there are lot of outliers and extreme values in each of the data set. First, we will replace them with NA using boxplot method and then with KNN imputation method, we will impute missing values which will remove most of the outliers from the data set. In Figure 2.6 we have plotted the boxplots for the same 14 variables after outlier removals. As it can be seen clearly the number of of outliers are less and extreme values are removed from the dataset. And now data is normally distributed.

(a) With Outlier (b) Without Outlier

Figure 2.1: Effect of Outliers

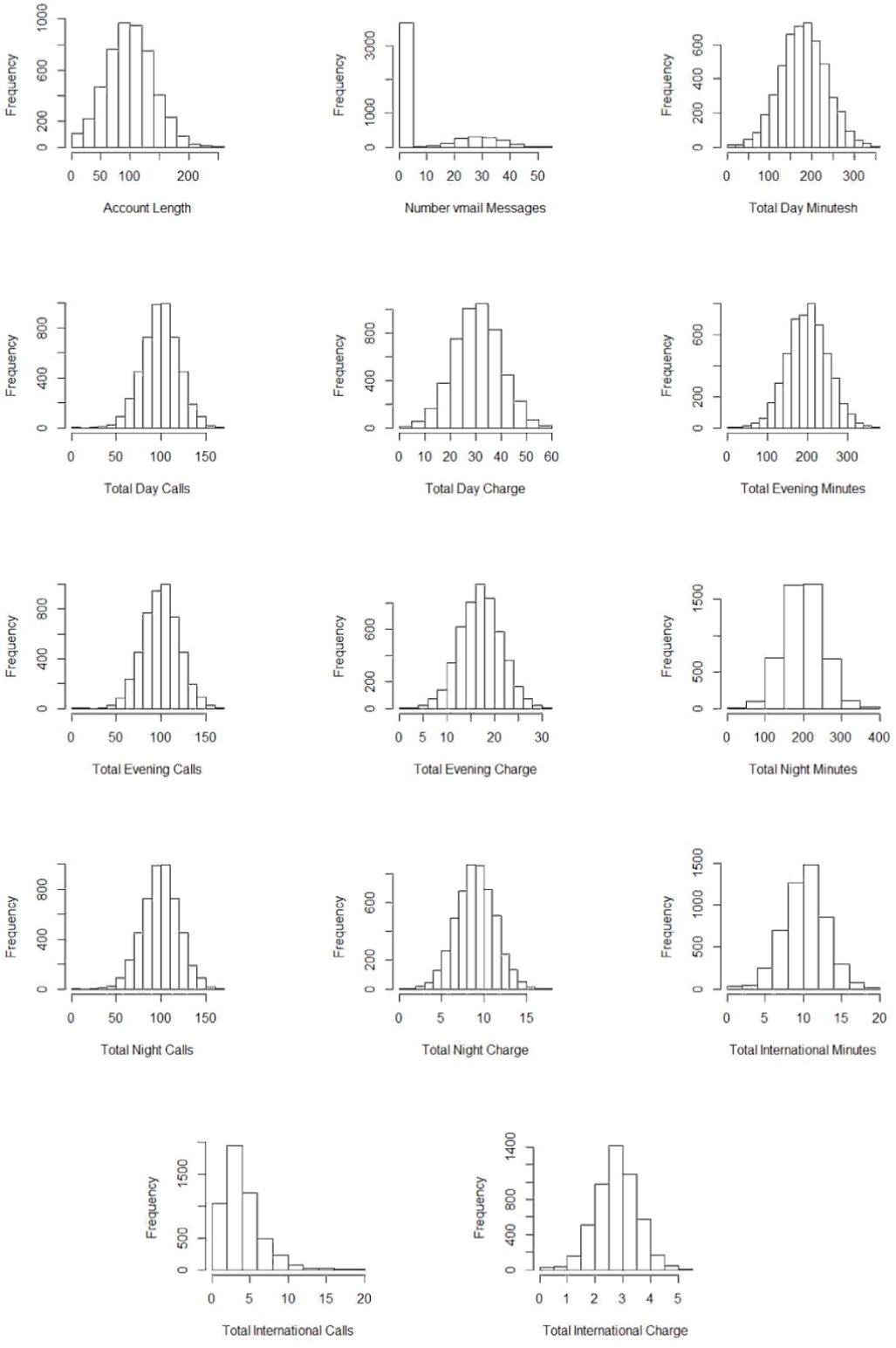


Figure 2.2: Histogram of Variables with Outliers

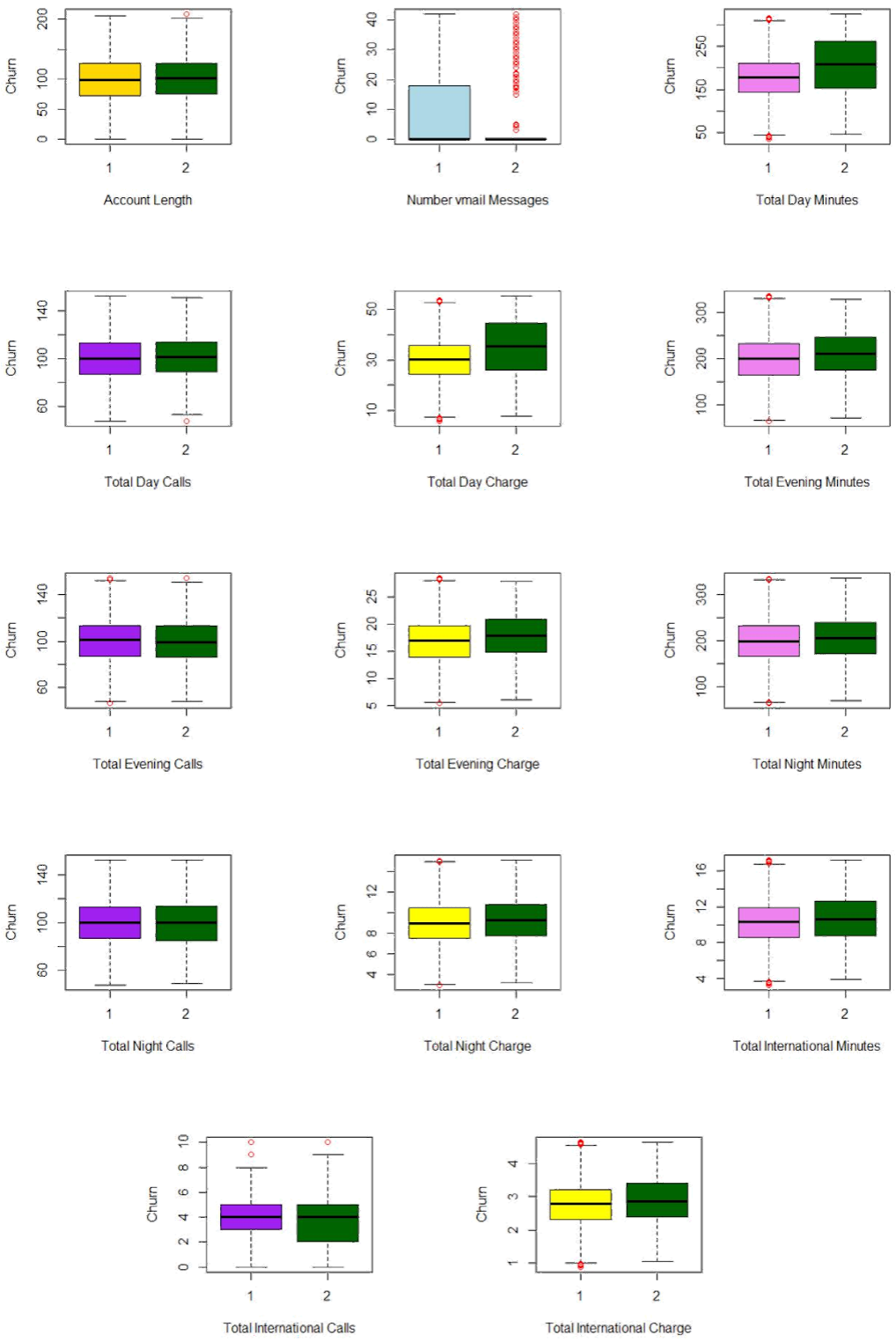
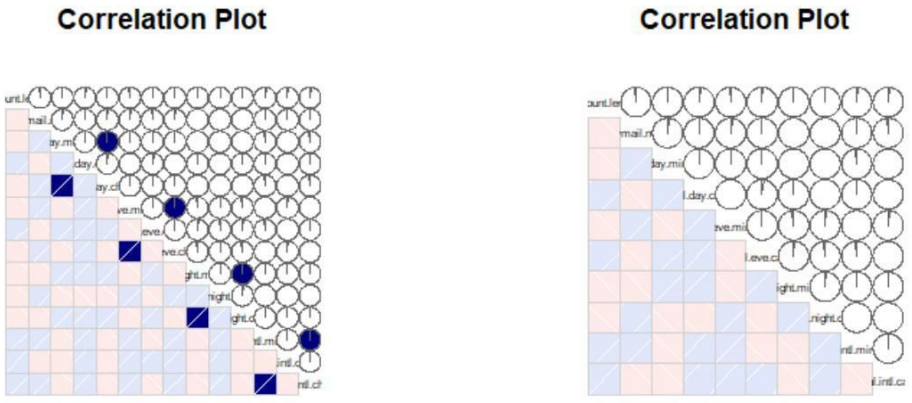


Figure 2.3: BoxPlot of Variables with Outlier

2.1.2 Feature Selection

Before performing any type of modeling we need to assess the importance of each predictor variable in our analysis. There is a possibility that many variables in our analysis are not important at all to the problem of classification. There are several methods of doing that. We have used the correlation analysis for continuous variables and chi-square test for the categorical variables.



(a) Before feature selection (b) After feature selection

Figure 2.5: Correlation Plot

The Figure 2.5a is correlation plot for Churn Reduction Data, it shows the correlation between two variables. The blue shade colour implies that two variables are positively correlated and orange shade implies negative correlation. As it can be clearly seen from figure total time variables is highly correlated with total charge as it should be. So, one of this variables can be dropped as they provide similar information and removing one of them will not effect model much.

For categorical variables, we used chi-square test of independence to check the dependence of variable with each other. If p value in chi-square test is less than 0.05, then we accept the variable for model otherwise reject it. The p-value implies dependence of variable on the target variable.

Therefore, following continuous variables can be removed after correlation analysis :

1. Numeric Variables:
    a. total day charge
    b. total eve charge
    c. total night charge
    d. total intl charge
2. Categorical Variables:
    a. area code
    b. phone number

2.1.3 Feature Scaling

Some variables range from 0 to 10 while other range from 0 to 1000. We need to normalise this, so that model should not be more prone towards the high value variables. We can do this either by standardization or normalization. Standardization is more suited for the data which is normally distributed. As for Churn Reduction data is not normally distributed. So we can't use standardization technique, Normalization is more suitable for such data set. After normalization all numeric data values will be between 0 and 1.

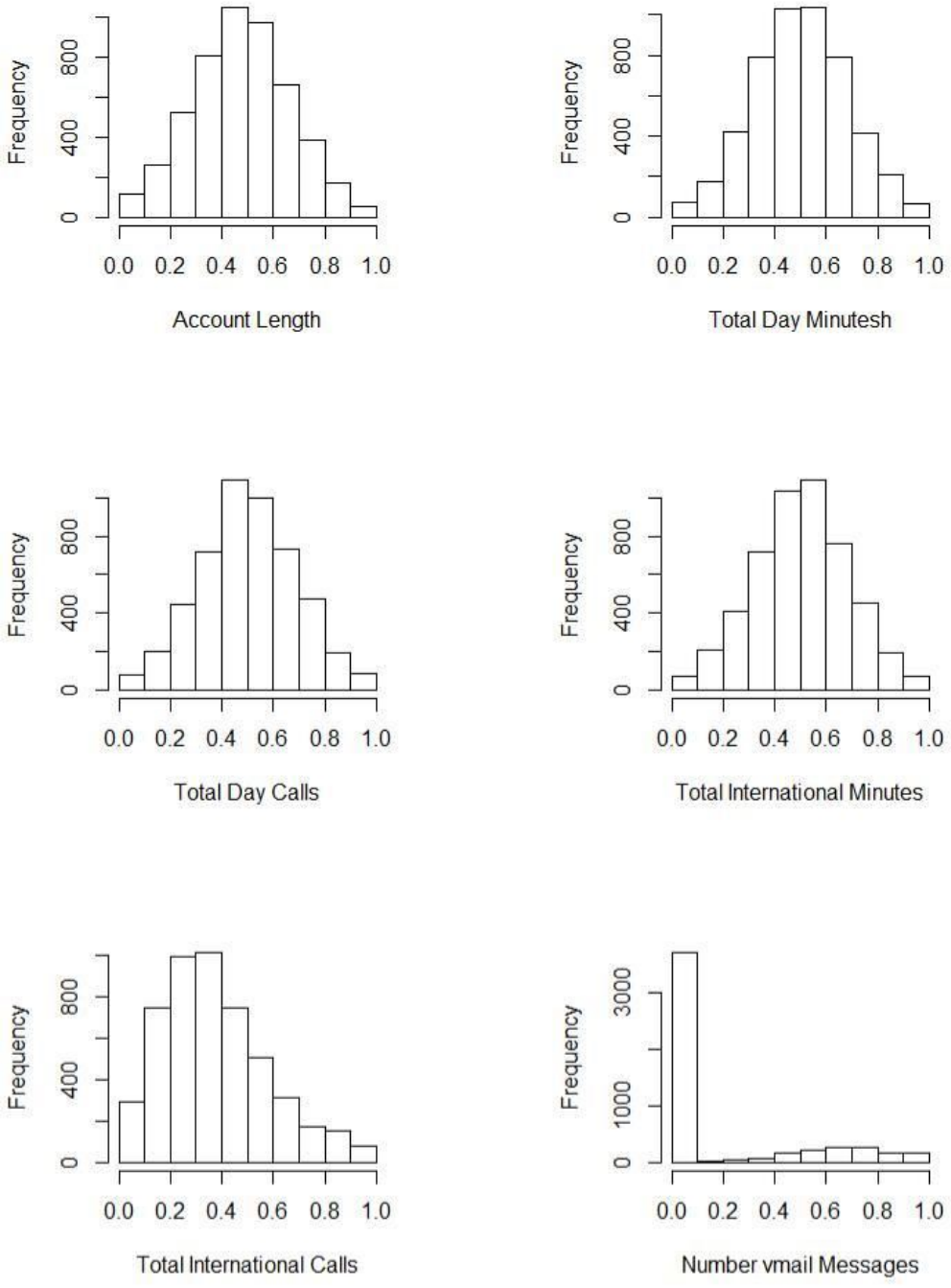


Figure 2.6: Histogram of Normalized Variables

2.2 Modeling

Model Selection

After preprocessing of data, we must proceed with model development. In this case, we have to predict whether customer will churn out or not. So this can be considered as binary classification problem, like if the customer has moved (1=yes; 0 = no). If the target variable is numeric or interval, then we must go for regression model. We can implement following models for data classification.

1. Decision Tree
2. Random Forest
3. Logistic Regression
4. KNN
5. Naive Bayes

2.2.1 Decision Tree

Decision tree is a type of supervised learning algorithm (having a predefined target variable) that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter/differentiator in input variables . Decision tree is a rule. Each branch connects nodes with “and” and multiple branches are connected by “or”. It can be used for classification and regression. It is a Supervised machine learning algorithm. Accept continuous and categorical variables as independent variables. Extremely easy to understand by the business users. We are using C5.0 model which is entropy based. When we applied this model on our train data, we got certain rules which is provided in a text file. You can also visualize the decision tree with the dot in python.

```
> confusionMatrix(ConfMatrix_DT)
Confusion Matrix and Statistics

   DT_Predictions
      1   2
  1 857   1
  2  60  81

               Accuracy : 0.9389
                 95% CI : (0.9223, 0.953)
    No Information Rate : 0.9179
    P-Value [Acc > NIR] : 0.007208

                  Kappa : 0.6948
 Mcnemar's Test P-Value : 1.118e-13

            Sensitivity : 0.9346
            Specificity : 0.9878
         Pos Pred Value : 0.9988
         Neg Pred Value : 0.5745
             Prevalence : 0.9179
         Detection Rate : 0.8579
   Detection Prevalence : 0.8589
      Balanced Accuracy : 0.9612

       'Positive' Class : 1

>
> #False Negative rate
> FNR = FN/FN+TP
Error: object 'FN' not found
>
> #Accuracy : 93.09%
> #FNR : 42.5
```

2.2.2 Random Forest

Random Forest or decision tree forests is an ensemble learning method for classification, regression and other tasks. Random Forest is a supervised learning algorithm. Like you can already see from its name, it creates a forest and makes it somehow random. The forest it builds, is an ensemble of Decision Trees, most of the time trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result .One big advantage of random forest is, that it can be used for both classification and regression problems, which form the majority of current machine learning systems. I will talk about random forest in classification, since classification is sometimes considered the building block of machine learning.We can see certain rules of random forest in the R code. The accuracy of the model is as follows.

```
> confusionMatrix(ConfMatrix_RF)
Confusion Matrix and Statistics

   RF_Predictions
      1   2
  1 851   7
  2  73  68

               Accuracy : 0.9199
                 95% CI : (0.9013, 0.936)
    No Information Rate : 0.9249
    P-Value [Acc > NIR] : 0.7486

                  Kappa : 0.5894
 Mcnemar's Test P-Value : 3.67e-13

            Sensitivity : 0.9210
            Specificity : 0.9067
         Pos Pred Value : 0.9918
         Neg Pred Value : 0.4823
             Prevalence : 0.9249
         Detection Rate : 0.8519
   Detection Prevalence : 0.8589
      Balanced Accuracy : 0.9138

       'Positive' Class : 1

>
> #False Negative rate
> FNR = FN/FN+TP
Error: object 'FN' not found
>
> #Accuracy = 91.79
> #FNR = 51.06
```

2.2.3 Logistic Regression:

Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables. it is the appropriate regression analysis to conduct when the dependent variable is binary cases where the dependent variable has more than two outcome categories may be analyzed in multinomial logistic regression or if the multiple categories are ordered, in ordinal logistic regression.

```
> ##Evaluate the performance of classification model
> ConfMatrix_logit = table(test$Churn, logit_Predictions)
>
> #False Negative rate
> FNR = FN/FN+TP
Error: object 'FN' not found
>
> #Accuracy: 87.18
> #FNR: 80.85
>
```

2.2.4 KNN Implementation:

K-Nearest Neighbours algorithm (KNN) is a non-parametric method used for classification and regression. KNN is a type of instance- based learning or lazy learning and simplest of all machine learning algorithms. Even with such simplicity, it can give highly competitive results. It is more widely used in classification problems in the industry. KNN fairs across all parameters of 23 considerations. It is commonly used for its easy of interpretation and low calculation time. K-NN is a lazy learner because it doesn't learn a discriminative function from the training data but "memorizes" the training dataset instead. Predictions are made for a new instance (x) by searching through the entire training set for the K most similar instances (the neighbours) and summarizing the output variable for those K instances. For regression this might be the mean output variable, in classification this might be the mode (or most common) class value.To determine which of the K instances in the training dataset are most similar to a new input a distance measure is used. For real-valued input variables, the most popular distance measure is Euclidean distance. Euclidean distance is calculated as the square root of the sum of the squared differences between a new point (x) and an existing point (xi) across all input attributes j.

```
> confusionMatrix(ConfMatrix_KNN)
Confusion Matrix and Statistics


KNN_Predictions   1   2
              1 840 132
              2  18   9

               Accuracy : 0.8498
                 95% CI : (0.8262, 0.8714)
    No Information Rate : 0.8589
    P-Value [Acc > NIR] : 0.8068

                  Kappa : 0.0647
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.97902
            Specificity : 0.06383
         Pos Pred Value : 0.86420
         Neg Pred Value : 0.33333
             Prevalence : 0.85886
         Detection Rate : 0.84084
   Detection Prevalence : 0.97297
      Balanced Accuracy : 0.52143

       'Positive' Class : 1

> Accuracy: 87.59
> #FNR: 80.14
```

2.2.5 Naive Bayes

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Naive Bayes model is easy to build and particularly 24 useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods. In machine learning we are often interested in selecting the best hypothesis (h) given data (d). In a classification problem, our hypothesis (h) may be the class to assign for a new data instance (d). One of the easiest ways of selecting the most probable hypothesis given the data that we have that we can use as our prior knowledge about the problem. Bayes' Theorem provides a way that we can calculate the probability of a hypothesis given our prior knowledge.

Bayes' Theorem is stated as:
P(h|d) = (P(d|h) * P(h)) / P(d)

```
Confusion Matrix and Statistics

          predicted
observed   1   2
       1 842  16
       2 107  34

               Accuracy : 0.8769
                 95% CI : (0.8549, 0.8966)
    No Information Rate : 0.9499
    P-Value [Acc > NIR] : 1

                  Kappa : 0.3046
 Mcnemar's Test P-Value : 4.857e-16

            Sensitivity : 0.8872
            Specificity : 0.6800
         Pos Pred Value : 0.9814
         Neg Pred Value : 0.2411
             Prevalence : 0.9499
         Detection Rate : 0.8428
   Detection Prevalence : 0.8589
      Balanced Accuracy : 0.7836

       'Positive' Class : 1

>
> #Accuracy: 87.59
> #FNR: 80.14
```

Conclusion

3.1 Model Evaluation

Model evaluation is done on basis of error metrics. Error metrics explain the performance of a model. An important aspect of error metrics is their capability to discriminate among model results. Simply, building a predictive model is not our final thing. Creating and selecting a model which gives high accuracy on out of sample data. Hence, it is crucial to check accuracy or other metric of the model prior to computing predicted values. In our data as we applied classification models we have error metrics like confusion matrix out of which we checked accuracy and false negative rate.

SL.	Model	accuracy	FNR
1.	Decision Tree	93.09	42.5
2.	Random Forest	91.79	51.06
3.	Logistic Regression	87.18	80.85
4.	KNN	85.38	60.86
5.	Naive Bayes	87.59	80.14

3.2 Model Selection

As it can be seen from all the screenshots that Decision Tree has highest Accuracy as compared to any other model and Random Forest performs better in the False Negative Rate. But there is no drastic change in their performance. So any one of them can be selected based on the customer requirement i.e. if the accuracy is more important for customer then we must select Decision Tree, but customer is interested in False Negative Rate then select Random Forest.

Appendix A

R Code :

```
rm(list = ls())

getwd()

setwd("/home/neha/home/Project_1")

x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest",
"unbalanced", "C50", "dummies", "e1071", "Information",
      "MASS", "rpart", "gbm", "ROSE", 'sampling', 'DataCombine',
'inTrees',"readr","class")

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

rm(x)

#Read the data
data_train = read.csv("Train_data.csv")

data_test = read.csv("Test_data.csv")

#Let's combine the train and test data for further process
churn_data = rbind(data_train, data_test)

############Explore the data#############
str(churn_data)

dim(churn_data)

churn_data$international.plan=as.factor(churn_data$international.p
lan)

churn_data$voice.mail.plan=as.factor(churn_data$voice.mail.plan)
churn_data$area.code=as.factor(churn_data$area.code)
churn_data$Churn=as.factor(churn_data$Churn)
churn_data$state=as.factor(churn_data$state)
```

```
#Missing Value Analysis
sum(is.na(churn_data))
# o/p : 0
#Hence we do not have any missing value in this dataset


#Data Manupulation; convert string categories into factor numeric
for(i in 1:ncol(churn_data)){

  if(class(churn_data[,i]) == 'factor'){

    churn_data[,i] = factor(churn_data[,i],
labels=(1:length(levels(factor(churn_data[,i])))))

  }
}



######################Outlier Analysis#########################

numeric_index = sapply(churn_data,is.numeric)

numeric_data = churn_data[,numeric_index]

cnames = colnames(numeric_data)

for (i in 1:length(cnames)) {
  assign(paste0("gn",i), ggplot(aes_string( y = (cnames[i]), x=
"Churn") , data = subset(churn_data)) +
           stat_boxplot(geom = "errorbar" , width = 0.5) +
           geom_boxplot(outlier.color = "red", fill = "grey",
outlier.shape = 20, outlier.size = 1, notch = FALSE)+
           theme(legend.position = "bottom")+
           labs(y = cnames[i], x= "Churn")+
           ggtitle(paste("Boxplot" , cnames[i])))
}

# Plotting plots together
gridExtra::grid.arrange(gn1, gn2,gn3, ncol=3)
gridExtra::grid.arrange(gn4,gn5,gn6, ncol=3)
```

```
gridExtra::grid.arrange(gn7,gn8,gn9, ncol =3)
gridExtra::grid.arrange(gn10,gn11, ncol =3 )

#Replace all outliers with NA and impute

for(i in cnames){
  val = churn_data[,i][churn_data[,i] %in%
boxplot.stats(churn_data[,i])$out]
  #print(length(val))
  churn_data[,i][churn_data[,i] %in% val] = NA
}

churn_data = knnImputation(churn_data, k = 3)

##########################Feature Selection###################

corrgram(churn_data[,numeric_index], order = F,
upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation
Plot")

## Chi-squared Test of Independence
factor_index = sapply(churn_data,is.factor)

factor_data = churn_data[,factor_index]

for (i in 1:6){
  print(names(factor_data)[i])
  print(chisq.test(table(factor_data$Churn,factor_data[,i])))
}

# Dimension Reduction
churn_data = subset(churn_data, select =
-c(total.day.charge,total.eve.charge,total.night.charge,total.intl
.charge,
                                              area.code,phone.number))

########################Feature Scaling#################
#Normality Check
cnames = colnames(churn_data[,sapply(churn_data,is.numeric)])

for(i in cnames){
```

```
  churn_data[,i] = (churn_data[,i] - min(churn_data[,i])) /
(max(churn_data[,i] - min(churn_data[,i])))
}

##################Model Development#############################

#Divide data into train and test using stratified sampling method
train.index = createDataPartition(churn_data$Churn, p = .80, list
= FALSE)
train = churn_data[ train.index,]
test  = churn_data[-train.index,]

#1.DECISION TREE CLASSIFIER
#Develop Model on training data

DT_model = C5.0(Churn ~., train, trials = 100, rules = TRUE)

#Summary of DT model
summary(DT_model)

#write rules into disk
write(capture.output(summary(DT_model)), "DTRules.txt")

#Lets predict for test cases
DT_Predictions = predict(DT_model, test[,-15], type = "class")

#Evaluate the performance of classification model
ConfMatrix_DT = table(test$Churn, DT_Predictions)

confusionMatrix(ConfMatrix_DT)

#False Negative rate
FNR = FN/FN+TP

#Accuracy : 93.09%
#FNR : 42.5

#2.Random Forest
RF_model = randomForest(Churn ~ ., train, importance = TRUE, ntree
= 500)
```

```
#Extract rules from random forest
#transform rf object to an inTrees' format
treeList = RF2List(RF_model)

#Extract rules
exec = extractRules(treeList, train[,-15])

#Visualize some rules
exec[1:2,]

#Make rules more readable:
readableRules = presentRules(exec, colnames(train))

readableRules[1:2,]

#Get rule metrics
ruleMetric = getRuleMetric(exec, train[,-15], train$Churn)

ruleMatric[1:2,]

#Predict test data using random forest model
RF_Predictions = predict(RF_model, test[,-15])

#Evaluate the performance of classification model
ConfMatrix_RF = table(test$Churn, RF_Predictions)

confusionMatrix(ConfMatrix_RF)

#False Negative rate
FNR = FN/FN+TP

#Accuracy = 91.79
#FNR = 51.06

#3.Logistic Regression

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

#summary of the model
summary(logit_model)
```

```
#predict using logistic regression
logit_Predictions = predict(logit_model, newdata = test, type =
"response")

#convert prob
logit_Predictions = ifelse(logit_Predictions > 0.5, 1, 0)


##Evaluate the performance of classification model
ConfMatrix_logit = table(test$Churn, logit_Predictions)

#False Negative rate
FNR = FN/FN+TP

#Accuracy: 87.18
#FNR: 80.85

#4.KNN Implementation
library(class)

#Predict test data
KNN_Predictions = knn(train[, 1:14], test[, 1:14], train$Churn, k
= 7)

#Confusion matrix
ConfMatrix_KNN = table(KNN_Predictions, test$Churn)
confusionMatrix(ConfMatrix_KNN)

#Accuracy
sum(diag(ConfMatrix_KNN))/nrow(test)

#False Negative rate
FNR = FN/FN+TP

#Accuracy = 85.38
#FNR = 60.86

#naive Bayes
library(e1071)

#Develop model
NB_model = naiveBayes(Churn ~ ., data = train)
```

```
#predict on test cases #raw
NB_Predictions = predict(NB_model, test[,1:14], type = 'class')



#Look at confusion matrix
ConfMatrix_NB = table(observed = test[,15], predicted =
NB_Predictions)
confusionMatrix(ConfMatrix_NB)

#Accuracy: 87.59
#FNR: 80.1
```

Python Code :

```
#import Libraries
import os
import pandas as pd
import numpy as np
from fancyimpute import KNN
import matplotlib.pyplot as plt
from sklearn.cross_validation import train_test_split
from sklearn import tree
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
import statsmodels.api as sm
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
import seaborn as sns
from scipy.stats import chi2_contingency
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
from sklearn.naive_bayes import GaussianNB


# In[2]:


#set working directory
os.chdir("/home/neha/home/Project_1/Python_Project1")



# In[3]:

os.getcwd()

# In[4]:

#Load train and test data
data_train = pd.read_csv("Train_data.csv")
data_test = pd.read_csv("Test_data.csv")

# In[5]:
```

```
#Let's combine the data for preprocessing
churn_data = data_train.append(data_test)
# In[6]:

churn_data.head()

# In[7]:

churn_data.info()

# # Exploratory Data Analysis

# In[8]:

## Convert Data into required types
churn_data['area code'] = churn_data['area code'].astype(str)
churn_data['number customer service calls'] = churn_data['number
customer service calls'].astype(str)


# In[9]:


for i in range(0, len(churn_data.columns)):
    if(churn_data.iloc[:,i].dtypes == 'object'):
        churn_data.iloc[:,i] =
pd.Categorical(churn_data.iloc[:,i])
        churn_data.iloc[:,i] = churn_data.iloc[:,i].cat.codes
        churn_data.iloc[:,i] =
churn_data.iloc[:,i].astype('object')


# In[10]:


object_cnames = []
numeric_cnames = []
target_cnames = []
for i in range(0, len(churn_data.columns)):
    if(churn_data.iloc[:,i].dtypes == 'object' and
churn_data.columns[i] != 'Churn'):
```

```
        object_cnames.append(churn_data.columns[i])
    elif(churn_data.iloc[:,i].dtypes == 'int64' or
churn_data.iloc[:,i].dtypes == 'float64'):
        numeric_cnames.append(churn_data.columns[i])
    else:
        target_cnames.append(churn_data.columns[i])



# In[ ]:


#Let's check if there is any missing value
churn_data.isnull().sum()
#There is no missing value


# In[ ]:


churn_data.info()


# # Outlier Analysis

# In[11]:


get_ipython().run_line_magic('matplotlib', 'inline')
plt.boxplot(churn_data['account length'])

# In[12]:


#Detect and impute outliers with NA
for i in numeric_cnames:
     #print(i)
    q75, q25= np.percentile(churn_data.loc[:,i], [75 ,25])
    iqr = q75 - q25

    min = q25 - (iqr*1.5)
```

```
    max = q75 + (iqr*1.5)

    #print(min)
    #print(max)

    churn_data.loc[churn_data[i] < min,:i] = np.nan
    churn_data.loc[churn_data[i] > max,:i] = np.nan


# In[13]:


missing_val = pd.DataFrame(churn_data.isnull().sum())


# In[14]:


#impute with KNN
churn_data = pd.DataFrame(KNN(k = 3).complete(churn_data), columns
= churn_data.columns)


# # Feature Selection

# In[15]:


df_corr = churn_data.loc[:, numeric_cnames]


# In[16]:


f, ax = plt.subplots(figsize=(10, 8))

corr = df_corr.corr()

sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool),
cmap=sns.diverging_palette(220, 10, as_cmap=True),
            square=True, ax=ax)
```

```
# In[17]:


for i in object_cnames:
      print(i)
      chi2, p, dof, ex =
chi2_contingency(pd.crosstab(churn_data['Churn'], churn_data[i]))
      print(p)


# In[18]:


churn_data = churn_data.drop(['total day charge', 'total eve
charge', 'total night charge', 'total intl charge',
                              'area code', 'phone number'],
axis=1 )
#as area code is also of no use


# In[19]:


numeric_cnames = ['account length','number vmail messages','total
day minutes','total day calls','total eve minutes',
                  'total eve calls','total night minutes','total
night calls','total intl minutes',
                  'total intl calls']


# In[20]:


#Normalisation

for i in numeric_cnames:
    print(i)
    churn_data[i] = (churn_data[i] -
np.min(churn_data[i]))/(np.max(churn_data[i]) -
```

```
np.min(churn_data[i]))


# # Model Development

# In[21]:


#Replace target categories with yes or no
churn_data['Churn'] = churn_data['Churn'].replace(0, 'No')
churn_data['Churn'] = churn_data['Churn'].replace(1, 'Yes')


# In[22]:


#Divide the data in train and test
X = churn_data.values[:,0:14]
Y = churn_data.values[:,14]

X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size = 0.2)


# In[23]:

#1.Decision Tree
DT_clf = tree.DecisionTreeClassifier(criterion =
'entropy').fit(X_train, Y_train)



# In[24]:


Y_Predict = DT_clf.predict(X_test)


# In[25]:
```

```
#Confusion Matrix
CM_DT = confusion_matrix(Y_test, Y_Predict)


# In[26]:


CM_DT = pd.crosstab(Y_test, Y_Predict)


# In[27]:


TN = CM_DT.iloc[0,0]
FN = CM_DT.iloc[1,0]
TP = CM_DT.iloc[1,1]
FP = CM_DT.iloc[0,1]


# In[28]:


accuracy_score(Y_test, Y_Predict)*100


# In[29]:


(FN*100)/(FN+TP)


# In[30]:


#For Decision Tree
#Accuracy : 91.8
#FNR : 35.65


# In[31]:
```

```
#2.Random Forest
RF_Model = RandomForestClassifier(n_estimators=500).fit(X_train,
Y_train)

# In[32]:


RF_Predictions = RF_Model.predict(X_test)


# In[33]:


CM_RF = confusion_matrix(Y_test, Y_Predict)


# In[34]:


CM_RF = pd.crosstab(Y_test, Y_Predict)

TN = CM_RF.iloc[0,0]
FN = CM_RF.iloc[1,0]
TP = CM_RF.iloc[1,1]
FP = CM_RF.iloc[0,1]

#Accuracy
#((TN+TP)*100)/(TN+FN+TP+FP)

#FNR
(FN*100)/(FN+TP)


# In[89]:


#For Random Forest
#Accuracy : 91.8
#FNR :35.65
```

```
# In[90]:


#3.KNN Implementation

KNN_Model = KNeighborsClassifier(n_neighbors = 9).fit(X_train,
Y_train)


# In[92]:

KNN_Predictions = KNN_Model.predict(X_test)


# In[96]:


CM = pd.crosstab(Y_test, KNN_Predictions)

TN = CM.iloc[0,0]
FN = CM.iloc[1,0]
TP = CM.iloc[1,1]
FP = CM.iloc[0,1]

#check accuracy of model
#accuracy_score(y_test, y_pred)*100
#((TN+TP)*100)/(TN+FN+TP+FP)

#FNR
(FN*100)/(FN+TP)


# In[97]:


#For KNN Implementation
#Accuracy : 85.2
#FNR : 92.46


# In[98]:
```

```
#4.Naive Bayes

NB_Model = GaussianNB().fit(X_train, Y_train)


# In[99]:


NB_Predictions = NB_Model.predict(X_test)


# In[103]:


CM = pd.crosstab(Y_test, NB_Predictions)

TN = CM.iloc[0,0]
FN = CM.iloc[1,0]
TP = CM.iloc[1,1]
FP = CM.iloc[0,1]

#check accuracy of model
#((TN+TP)*100)/(TN+FN+TP+FP)

#FNR
(FN*100)/(FN+TP)

#For Naive Bayes
#Accuracy : 85.1
#FNR : 63.69
```