CUSTOMER ATTRITION

Table of Contents

Introduction

Customer Attrition

Customer Attrition means that a subscriber or a user ends his or her relationship with a company. Internet service providers, telephone service companies, insurance companies and alarm monitoring companies often use customer attrition rates analysis as one of the key metrices.

The reason this is important is that the cost of acquiring a new customer is far more than retaining an existing one. Customer Attrition can be categorized as – Voluntary and Involuntary. Voluntary attrition is due to the decision taken by the customer to switch to another company or service provider. However, involuntary attrition is due to circumstances beyond the customer's control. Voluntary Attrition is more important because it occurs due to factors which companies control.

Data Source

The data is taken from Kaggle.

Problem

The problem at hand is one of classification. We are interested in knowing whether the customer is retained or not depending on the customer specific factors.

Data Preparation

Data Structure

Variable	Number	Unique Values
customerID	7043	
gender	2	(Male, Female)
SeniorCitizen	2	(Yes - 1, No - 0)
Partner	2	(Yes, No)
Dependents	2	(Yes, No)
tenure	73	(Months)
PhoneService	2	(Yes, No)
MultipleLines	3	(Yes, No, No Phone service)
InternetService	3	(DSL, Fiber Optic, No)
OnlineSecurity	3	(Yes, No, No Internet service)
OnlineBackup	3	(Yes, No, No Internet service)
DeviceProtection	3	(Yes, No, No Internet service)
TechSupport	3	(Yes, No, No Internet service)
StreamingTV	3	(Yes, No, No Internet service)
StreamingMovies	3	(Yes, No, No Internet service)
Contract	3	(Month-to-Month, One year, Two year)
PaperlessBilling	2	(Yes, No)
PaymentMethod	4	(Electronic check, Mailed check, Bank transfer (automatic), Credit card (automatic))
MonthlyCharges	1585	($ Monthly Expenditure)
TotalCharges	6531	($ Monthly/Yearly/Two-Yearly Expenditure)
Churn	2	(Yes, No)

Data Transformation

1. Variables 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies' had an unique value ‘No internet service’. This was replaced by the value ‘No’.

2. Values in the variable 'Senior Citizen were changed from [1,0] to ['Yes,'No'].
3. 'Tenure' had values ranging from 1 to 72 months. This is converted into annual buckets "<=12", "<=24", "<=36", "<=48", "<=60", ">60".

This resulted into the following distribution between categorical and numerical variables:

Categorical Features	
gender	DeviceProtection
SeniorCitizen	TechSupport
Partner	StreamingTV
Dependents	StreamingMovies
PhoneService	Contract
MultipleLines	PaperlessBilling
InternetService	PaymentMethod
OnlineSecurity	tenure_bucket
OnlineBackup	

Numereical Features
MonthlyCharges
TotalCharges
Churn

The categorical features cannot be used to fit a model. In order to fit models, they need to be converted into numerical values. This is done with the help of 'One Hot Encoding' technique. A one hot encoding is a representation of categorical variables as binary vectors. This first requires that the categorical values be mapped to integer values. Then, each integer value is represented as a binary vector that is all zero values except the index of the integer, which is marked with a 1.

After all the above steps are completed, the data can be used for fitting the different types of models. The data is divided into training (80%) and validation (20%) to test the efficacy of the model.

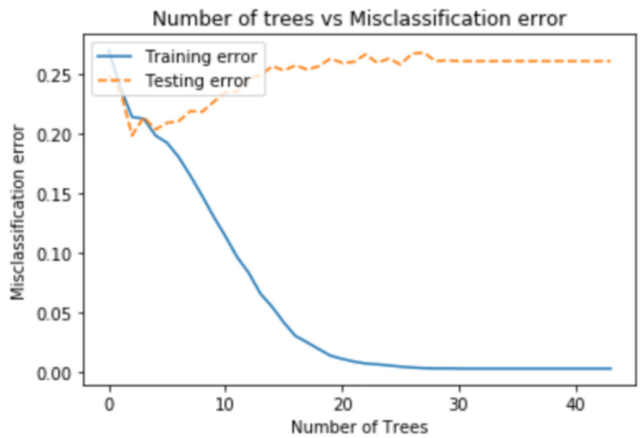
Models

The models used for classification are:

- Decision Tree
- Random Forests
- Logistic Regression
- Naïve Bayes
- K – Nearest Neighbor
- XGBoost

Decision Tree

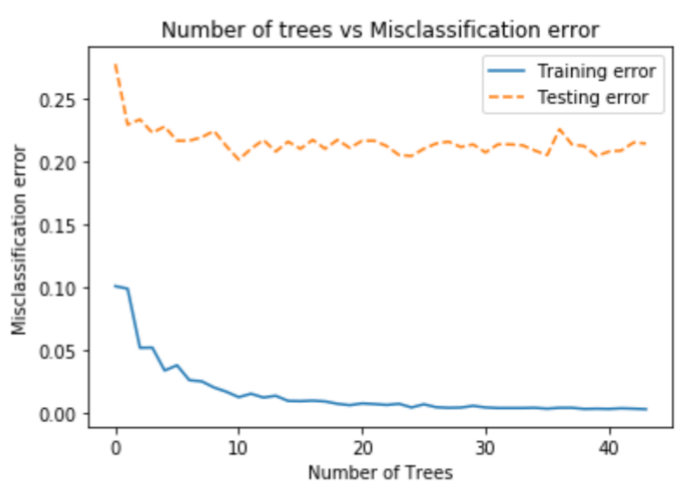
Different decision trees with varying levels of maximum tree depth were fitted. The graph below shows the values of training and test errors at different maximum tree depths. A detailed list of the observed rate of error for test and validation data along with the AUC is available in Appendix – I.



It was observed that the minimum error of 0.21% in the training was observed at a tree depth of 25. However, for the validation data, the minimum error of 21.75% in the training was observed at a tree depth of 7.

Random Forest

Different random forests with varying levels of maximum number of tree in the forest were fitted. The graph below shows the values of training and test errors at different maximum number of tree in the forest. A detailed list of the observed rate of error for test and validation data along with the AUC is available in Appendix – II.



It was observed that the minimum error of 0.23% in the training was observed at a maximum number of tree in the forest of 44. However, for the validation data, the minimum error of 21.68% in the training was observed at a maximum number of tree in the forest of 44.

Logistic Regression

The training data had a classification error of 19.4% while the validation data had a classification of 20.68%.

The coefficients of various variables for the model were:

Variable Name	Coefficient
gender - Female	-0.10153092
gender - Male	-0.1314278
SeniorCitizen - Yes	-0.33240085
SeniorCitizen - No	0.09944213
Partner - Yes	-0.04080036
Partner - No	-0.19215836
Dependents - Yes	0.00513249
Dependents - No	-0.23809121
PhoneService - Yes	0.01861081
PhoneService - No	-0.25156953
MultipleLines - No phone service	-0.2983611
MultipleLines - Yes	0.01861081
MultipleLines - No	0.04679157
InternetService - DSL	-0.26147826
InternetService - No	0.32398709
InternetService - Fiber optic	-0.29546755
OnlineSecurity - Yes	0.05300691
OnlineSecurity - No	-0.28596563
OnlineBackup - Yes	-0.08087003
OnlineBackup - No	-0.15208869
DeviceProtection - Yes	-0.06627048
DeviceProtection - No	-0.16668824

Variable Name	Coefficient
TechSupport - Yes	0.05226872
TechSupport - No	-0.28522744
StreamingTV - Yes	-0.20836366
StreamingTV - No	-0.02459506
StreamingMovies - Yes	-0.23355227
StreamingMovies - No	0.00059355
Contract - Two year	0.32815161
Contract - One year	-0.21347708
Contract - Month-to-month	-0.34763325
PaperlessBilling - Yes	-0.32984375
PaperlessBilling - No	0.09688503
PaymentMethod - Credit card automatic	-0.13900561
PaymentMethod - Bank transfer automatic	-0.18198373
PaymentMethod - Electronic check	0.27832232
PaymentMethod - Mailed check	-0.1902917
tenure_bucket - >60	0.21891044
tenure_bucket - 12-24	-0.21824872
tenure_bucket - 24-48	-0.20020689
tenure_bucket - 48-60	0.00820551
tenure_bucket - 0-12	-0.04161907
MonthlyCharges	0.01944051
TotalCharges	-0.0003809

Naïve Bayes

The training data had a classification error of 24.8% while the validation data had a classification of 24.88%.

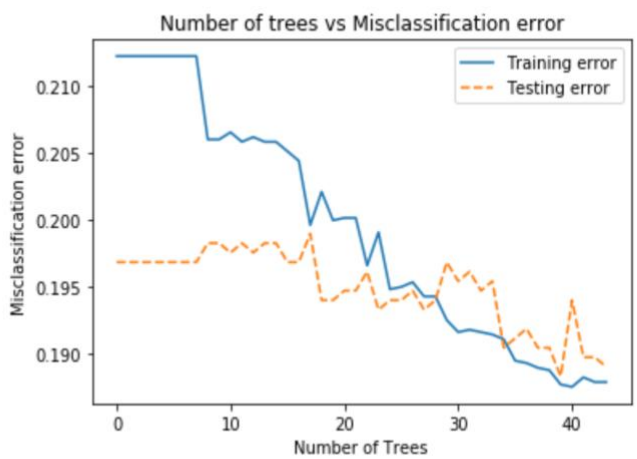
K – nearest Neighbor

KNN is fitted with values of k from [3, 4, 5, 7]. It is observed that the training errors increase from 13.58% to 17.83% with increase in value of k. However, the validation errors change abruptly. The table below shows the reported errors:

Model	Train Error	Validation Error
KNN - 3	13.58%	25.52%
KNN - 4	16.78%	24.38%
KNN - 5	16.57%	25.52%
KNN - 7	17.83%	23.88%

XGBoost

Different XGBoost models with varying levels of number of estimators were fitted. The graph below shows the values of training and test errors at different number of estimators. A detailed list of the observed rate of error for test and validation data along with the AUC is available in Appendix – III.



It was observed that the minimum error of 18.46% in the training was observed at a tree depth of 44. However, for the validation data, the minimum error of 20.19% in the training was observed at a tree depth of 37.

Conclusion

The error rates of test and validation data along with the AUC is:

Model	Train		Validation	
	Error	**AUC**	**Error**	**AUC**
Decision Tree	0.27%	99.52%	20.90%	69.77%
Random Forest	0.30%	99.56%	20.68%	69.81%
Logistic	19.73%	70.27%	20.18%	71.25%
Naïve Bayes	24.87%	75.89%	25.37%	76.98%
KNN - 3	13.58%	80.12%	25.52%	65.10%
KNN - 4	16.78%	71.21%	24.38%	61.92%
KNN - 5	16.57%	75.25%	25.52%	64.11%
KNN - 7	17.83%	73.16%	23.88%	65.72%
XGBoost	18.40%	72.47%	20.47%	70.73%

It can be seen that Logistic Regression has the minimum value of Validation error. XGBoost comes in close second. Logistic Regression also has the second highest AUC. Naïve Bayes has the highest AUC but at the same time has the second highest level of error.

We conclude that Logistic Regression model provides the best fit to the data in hand.

Code

Processing

```
#Importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn import cross_validation

#Load Data
t = pd.read_csv("/Users/neerajjeswani/Desktop/Telco/TCC.csv")
t.head()

#Overall Summary
print ("Rows     : " ,t.shape[0])
print ("Columns  : " ,t.shape[1])
print ("Features : \n" ,t.columns.tolist())
print ("Missing values :  ", t.isnull().sum().values.sum())
print ("Unique values in each Feature :  \n",t.nunique())

print(type(t["TotalCharges"][1]))

#Replacing Blanks
t['TotalCharges'] = t["TotalCharges"].replace(" ",np.nan)
t = t[t["TotalCharges"].notnull()]
t = t.reset_index()[t.columns]

#Converting data type
t["TotalCharges"] = t["TotalCharges"].astype(float)

print(type(t["TotalCharges"][1]))

#Finding at Unique Values
for col in t:
   print(col, t[col].unique())

#Replacing 'No Internet Service'
to_replace = [ 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection','TechSupport','StreamingTV', 'StreamingMovies']

for col in to_replace :
   t[col]  = t[col].replace({'No internet service' : 'No'})

#Changing Senior Citizen variable to 'Yes/No'
t["SeniorCitizen"] = t["SeniorCitizen"].replace({1:"Yes",0:"No"})

#Bucketting Tenure
```

```
def buckets(df) :

    if df["tenure"] <= 12 :
        return "0-12"
    elif (df["tenure"] > 12) & (df["tenure"] <= 24 ):
        return "12-24"
    elif (df["tenure"] > 24) & (df["tenure"] <= 48) :
        return "24-48"
    elif (df["tenure"] > 48) & (df["tenure"] <= 60) :
        return "48-60"
    elif df["tenure"] > 60 :
        return ">60"

t["tenure_bucket"] = t.apply(lambda t:buckets(t),axis = 1)
t.drop("tenure", axis=1, inplace=True)

#Differentiating Columns into Categorical & Numerical
Id_col     = ['customerID']
target_col = ["Churn"]
cat_cols   = t.nunique()[t.nunique() < 6].keys().tolist()
cat_cols   = [x for x in cat_cols if x not in target_col]
num_cols   = [x for x in t.columns if x not in cat_cols + Id_col]

df1 = t[cat_cols]
df2 = t[num_cols]
df=pd.merge(left=df1, right=df2, left_index=True, right_index=True)

df["Churn"] = df["Churn"].replace({"Yes":1,"No":0})

#Converting categorical into numerical
cols = df.columns

labels = []

for i in range(0,17):
    train = df[cols[i]].unique()
    labels.append(list(set(train)))

cats = []

for i in range(0, 17):
    #Label encode
    label_encoder = LabelEncoder()
    label_encoder.fit(labels[i])
    feature = label_encoder.transform(df.iloc[:,i])
    feature = feature.reshape(df.shape[0], 1)
    #One hot encode
    onehot_encoder = OneHotEncoder(sparse=False,n_values=len(labels[i]))
    feature = onehot_encoder.fit_transform(feature)
    cats.append(feature)

# Make a 2D array from a list of 1D arrays
encoded_cats = np.column_stack(cats)
```

```
df = np.concatenate((encoded_cats,df.iloc[:,17:].values),axis=1)

#Creating X & Y dataframes
X = df[:,0:44]
Y = df[:,44]

#Splitting into Train and Test Data
X_train, X_val, Y_train, Y_val = cross_validation.train_test_split(X, Y, test_size=.2)

#Saving Data
np.savetxt("/Users/neerajjeswani/Desktop/Telco/X_train.csv", X_train, delimiter=",")
np.savetxt("/Users/neerajjeswani/Desktop/Telco/X_val.csv", X_val, delimiter=",")
np.savetxt("/Users/neerajjeswani/Desktop/Telco/Y_train.csv", Y_train, delimiter=",")
np.savetxt("/Users/neerajjeswani/Desktop/Telco/Y_Val.csv", Y_val, delimiter=",")
```

Models

```
#Import Libraries
import numpy as np
import pandas as pd
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

#Load Data Set
X_train = np.genfromtxt("/Users/neerajjeswani/Desktop/Telco/X_train.csv", delimiter=",")
X_val = np.genfromtxt("/Users/neerajjeswani/Desktop/Telco/X_val.csv", delimiter=",")
Y_train = np.genfromtxt("/Users/neerajjeswani/Desktop/Telco/Y_train.csv", delimiter=",")
Y_val = np.genfromtxt("/Users/neerajjeswani/Desktop/Telco/Y_Val.csv", delimiter=",")

####### DECISION TREE #######

##Tables for classification errors
error_val =[]
error_train =[]
tree_depth =[]
auc_train = []
auc_val = []

min_error_train = 1
tree_depth_train = 0
min_error_val = 1
tree_depth_val = 0

for i in range(1,45):
   d_tree = DecisionTreeClassifier(criterion = 'entropy', random_state = 7, max_depth = i)
   d_tree.fit(X_train, Y_train)
```

```
    ##Training Classification error
    Y_pred = d_tree.predict(X_train)
    error = 1 - accuracy_score(Y_train,Y_pred)
    error_train.append(error)
    model_roc_auc = roc_auc_score(Y_train,Y_pred)
    auc_train.append(model_roc_auc)

    if (min_error_train>error):
      min_error_train=error
      tree_depth_train = i

    ##Validation Classification error
    Y_pred = d_tree.predict(X_val)
    error = 1 - accuracy_score(Y_val,Y_pred)
    error_val.append(error)
    model_roc_auc = roc_auc_score(Y_val,Y_pred)
    auc_val.append(model_roc_auc)

    if (min_error_val>error):
      min_error_val=error
      tree_depth_val = i

    tree_depth.append(i)

#AUC and Confusion Matrix for the tree with the minimum error for validation data set
d_tree = DecisionTreeClassifier(criterion = 'entropy', random_state = 7, max_depth = 6)
d_tree.fit(X_train, Y_train)

Y_pred = d_tree.predict(X_train)
error = 1 - accuracy_score(Y_train,Y_pred)
error_train.append(error)
model_roc_auc = roc_auc_score(Y_train,Y_pred)
confusion_matrix(Y_train,Y_pred)

Y_pred = d_tree.predict(X_val)
error = 1 - accuracy_score(Y_val,Y_pred)
error_train.append(error)
model_roc_auc = roc_auc_score(Y_val,Y_pred)
confusion_matrix(Y_val,Y_pred)

####### RANDOM FOREST #######

from sklearn.ensemble import RandomForestClassifier

##Tables for classification errors
error_train =[]
error_val =[]
auc_train = []
auc_val = []

min_error_train = 1
tree_depth_train = 0
min_error_val = 1
```

```
tree_depth_val = 0

for k in range(1,45):

    rf = RandomForestClassifier(criterion='entropy', n_estimators=k, max_features = 'sqrt')
    rf.fit(X_train, np.ravel(Y_train,order='C'))

    ##Training Classification error
    Y_pred = rf.predict(X_train)
    error = round(1 - accuracy_score(Y_train,Y_pred),4)
    error_train.append(error)
    model_roc_auc = roc_auc_score(Y_train,Y_pred)
    auc_train.append(model_roc_auc)

    if (min_error_train>error):
        min_error_train=error
        tree_depth_train = i

    ##Validation Classification error
    Y_pred = rf.predict(X_val)
    error = round(1 - accuracy_score(Y_val,Y_pred),4)
    error_val.append(error)
    model_roc_auc = roc_auc_score(Y_val,Y_pred)
    auc_val.append(model_roc_auc)

    if (min_error_val>error):
        min_error_val=error
        tree_depth_val = i

#AUC and Confusion Matrix for the tree with the minimum error for validation data set
rf = RandomForestClassifier(criterion='entropy', n_estimators=44, max_features = 'sqrt')
rf.fit(X_train, np.ravel(Y_train,order='C'))

Y_pred = rf.predict(X_train)
error = round(1 - accuracy_score(Y_train,Y_pred),4)
error_train.append(error)
model_roc_auc = roc_auc_score(Y_train,Y_pred)
confusion_matrix(Y_train,Y_pred)

Y_pred = rf.predict(X_val)
error = round(1 - accuracy_score(Y_val,Y_pred),4)
error_val.append(error)
model_roc_auc = roc_auc_score(Y_val,Y_pred)
confusion_matrix(Y_val,Y_pred)

####### LOGISTIC REGRESSION #######

from sklearn.linear_model import LogisticRegression

LogReg = LogisticRegression()
LogReg.fit(X_train, Y_train)
Log = LogReg.fit(X_train, Y_train)
y_pred = LogReg.predict(X_train)
```

```
error = 1 - LogReg.score(X_train, Y_train)
print(error)
model_roc_auc = roc_auc_score(Y_train,y_pred)
print(model_roc_auc)
confusion_matrix(Y_train,y_pred)

y_pred = LogReg.predict(X_val)

error = 1 - LogReg.score(X_val, Y_val)
print(error)
model_roc_auc = roc_auc_score(Y_val,y_pred)
print(model_roc_auc)
confusion_matrix(Y_val,y_pred)

coefficients_log = LogReg.coef_

####### NAIVE BAYES #######

from sklearn.naive_bayes import GaussianNB

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

y_pred = model.predict(X_train)

error = 1- model.score(X_train, Y_train)
print(error)
model_roc_auc = roc_auc_score(Y_train,y_pred)
print(model_roc_auc)
confusion_matrix(Y_train,y_pred)

y_pred = model.predict(X_val)
error = 1- model.score(X_val, Y_val)
print(error)
model_roc_auc = roc_auc_score(Y_val,y_pred)
print(model_roc_auc)
confusion_matrix(Y_val,y_pred)

####### K NEAREST NEIGHBOR #######

from sklearn.neighbors import KNeighborsClassifier

neighbor = [3,4,5,7]

for i in neighbor:
  knn = KNeighborsClassifier(n_neighbors=i)
  knn.fit(X_train, Y_train)

  Y_pred = knn.predict(X_train)
  error = 1 - accuracy_score(Y_train,Y_pred)
  print(error)
  model_roc_auc = roc_auc_score(Y_train,Y_pred)
```

```
    print(model_roc_auc)
    confusion_matrix(Y_train,Y_pred)

    Y_pred = knn.predict(X_val)
    error = 1 - accuracy_score(Y_val,Y_pred)
    print(error)
    model_roc_auc = roc_auc_score(Y_val,Y_pred)
    print(model_roc_auc)
    confusion_matrix(Y_val,Y_pred)

####### XGBOOST #######

from xgboost import XGBClassifier

error_val =[]
error_train =[]
tree_depth =[]
auc_train = []
auc_val = []

min_error_train = 1
tree_depth_train = 0
min_error_val = 1
tree_depth_val = 0

for i in range(1,45):
    clf =XGBClassifier(n_estimators=i,seed=777)

    clf.fit(X_train, Y_train)

    Y_pred = clf.predict(X_train)
    error = 1 - accuracy_score(Y_train,Y_pred)
    error_train.append(error)
    model_roc_auc = roc_auc_score(Y_train,Y_pred)
    auc_train.append(model_roc_auc)

    if (min_error_train>error):
        min_error_train=error
        tree_depth_train = i

    ##Validation Classification error
    Y_pred = clf.predict(X_val)
    error = 1 - accuracy_score(Y_val,Y_pred)
    error_val.append(error)
    model_roc_auc = roc_auc_score(Y_val,Y_pred)
    auc_val.append(model_roc_auc)

    if (min_error_val>error):
        min_error_val=error
        tree_depth_val = i

    tree_depth.append(i)
```

```
#AUC and Confusion Matrix for the tree with the minimum error for validation data set
clf =XGBClassifier(n_estimators=29,seed=777)

clf.fit(X_train, Y_train)

Y_pred = clf.predict(X_train)
error = 1 - accuracy_score(Y_train,Y_pred)
error_train.append(error)
model_roc_auc = roc_auc_score(Y_train,Y_pred)
confusion_matrix(Y_train,Y_pred)

Y_pred = clf.predict(X_val)
error = 1 - accuracy_score(Y_val,Y_pred)
error_val.append(error)
model_roc_auc = roc_auc_score(Y_val,Y_pred)
confusion_matrix(Y_val,Y_pred)
```

Appendix – I : Decision Tree Rate of Error and AUC

Number of Trees	Train		Validation	
	Error	AUC	Error	AUC
1	26.45%	50.00%	27.08%	50.00%
2	23.61%	71.88%	24.52%	71.14%
3	20.84%	65.87%	22.03%	64.93%
4	20.69%	70.33%	21.96%	69.27%
5	19.50%	74.18%	21.82%	71.43%
6	18.65%	73.21%	21.82%	69.53%
7	17.62%	74.77%	21.75%	69.66%
8	16.57%	77.14%	22.53%	69.54%
9	15.04%	78.74%	23.10%	68.49%
10	13.99%	80.12%	22.46%	69.34%
11	11.89%	82.66%	23.95%	67.49%
12	9.76%	86.61%	24.80%	67.98%
13	8.04%	88.84%	25.23%	66.78%
14	6.42%	91.29%	25.23%	67.27%
15	5.26%	93.20%	25.09%	67.45%
16	4.36%	94.24%	26.01%	66.41%
17	3.25%	96.02%	25.16%	67.90%
18	2.51%	96.40%	25.94%	67.20%
19	1.90%	97.18%	25.73%	66.85%
20	1.33%	98.23%	26.15%	66.64%
21	1.00%	98.46%	26.15%	66.39%
22	0.68%	98.87%	26.01%	66.90%
23	0.46%	99.32%	25.80%	67.38%
24	0.32%	99.44%	25.59%	67.19%
25	0.21%	99.62%	25.09%	67.78%
26	0.21%	99.62%	25.09%	67.78%
27	0.21%	99.62%	25.09%	67.78%
28	0.21%	99.62%	25.09%	67.78%
29	0.21%	99.62%	25.09%	67.78%
30	0.21%	99.62%	25.09%	67.78%
31	0.21%	99.62%	25.09%	67.78%
32	0.21%	99.62%	25.09%	67.78%
33	0.21%	99.62%	25.09%	67.78%
34	0.21%	99.62%	25.09%	67.78%
35	0.21%	99.62%	25.09%	67.78%
36	0.21%	99.62%	25.09%	67.78%
37	0.21%	99.62%	25.09%	67.78%
38	0.21%	99.62%	25.09%	67.78%
39	0.21%	99.62%	25.09%	67.78%
40	0.21%	99.62%	25.09%	67.78%
41	0.21%	99.62%	25.09%	67.78%
42	0.21%	99.62%	25.09%	67.78%
43	0.21%	99.62%	25.09%	67.78%
44	0.21%	99.62%	25.09%	67.78%

Appendix – II : Random Forest Rate of Error and AUC

Number of Trees	Train		Validation	
	Error	AUC	Error	AUC
1	10.29%	86.40%	29.21%	63.22%
2	9.35%	83.38%	27.08%	59.24%
3	5.30%	92.81%	25.37%	67.01%
4	5.65%	90.00%	24.38%	63.90%
5	2.74%	95.99%	24.31%	67.49%
6	3.61%	93.72%	24.24%	64.16%
7	2.31%	96.73%	23.81%	68.25%
8	2.72%	95.31%	22.39%	67.57%
9	1.53%	97.71%	23.31%	68.42%
10	1.81%	96.94%	23.31%	67.35%
11	1.53%	97.76%	23.31%	68.42%
12	1.33%	97.82%	23.24%	66.57%
13	1.14%	98.26%	23.95%	67.24%
14	1.07%	98.33%	22.10%	68.01%
15	0.80%	98.83%	23.60%	67.90%
16	1.21%	97.91%	23.03%	66.80%
17	0.91%	98.72%	22.81%	68.76%
18	0.92%	98.45%	22.53%	67.89%
19	0.62%	99.10%	22.53%	69.12%
20	0.71%	98.81%	24.02%	66.04%
21	0.64%	99.03%	22.53%	69.37%
22	0.60%	99.03%	23.24%	67.15%
23	0.60%	99.09%	23.45%	68.08%
24	0.60%	99.03%	23.24%	67.81%
25	0.44%	99.38%	22.39%	69.47%
26	0.52%	99.13%	22.17%	68.30%
27	0.46%	99.30%	23.45%	67.42%
28	0.53%	99.14%	22.81%	67.44%
29	0.27%	99.63%	22.67%	68.28%
30	0.39%	99.39%	23.03%	67.96%
31	0.30%	99.60%	22.46%	69.26%
32	0.36%	99.50%	22.96%	68.09%
33	0.27%	99.69%	23.67%	67.35%
34	0.34%	99.49%	22.81%	68.10%
35	0.36%	99.54%	22.32%	69.27%
36	0.43%	99.43%	22.81%	68.19%
37	0.27%	99.69%	23.17%	67.86%
38	0.36%	99.52%	22.39%	68.48%
39	0.25%	99.62%	22.67%	68.78%
40	0.34%	99.56%	23.10%	68.07%
41	0.30%	99.58%	23.38%	67.55%
42	0.23%	99.71%	21.75%	69.50%
43	0.27%	99.58%	22.46%	68.76%
44	0.39%	99.37%	22.53%	68.63%

Appendix – III : XGBoost Rate of Error and AUC

Number of Trees	Train		Validation	
	Error	AUC	Error	AUC
1	20.69%	66.95%	21.82%	66.23%
2	20.69%	66.95%	21.82%	66.23%
3	20.69%	66.95%	21.82%	66.23%
4	20.69%	66.95%	21.82%	66.23%
5	20.69%	66.95%	21.82%	66.23%
6	20.69%	66.95%	21.82%	66.23%
7	20.71%	66.56%	21.39%	66.52%
8	20.71%	66.56%	21.39%	66.52%
9	20.30%	67.65%	21.11%	67.38%
10	20.18%	67.67%	21.39%	66.60%
11	20.30%	67.65%	21.11%	67.38%
12	20.30%	67.65%	21.11%	67.38%
13	20.18%	67.67%	21.39%	66.60%
14	20.12%	67.92%	21.32%	66.73%
15	20.12%	67.92%	21.32%	66.73%
16	20.05%	68.06%	21.32%	66.73%
17	19.54%	70.32%	21.18%	68.15%
18	19.96%	68.44%	21.18%	67.08%
19	19.48%	70.44%	21.11%	68.28%
20	19.29%	70.41%	20.90%	68.76%
21	19.32%	70.40%	21.18%	68.15%
22	19.40%	70.29%	20.97%	68.55%
23	19.27%	70.72%	20.75%	69.02%
24	19.16%	71.05%	20.61%	69.53%
25	19.20%	70.79%	20.75%	69.10%
26	19.09%	71.44%	20.54%	69.66%
27	19.06%	71.34%	20.33%	69.97%
28	18.95%	71.60%	20.33%	70.06%
29	18.95%	71.50%	20.40%	69.84%
30	19.02%	71.49%	20.26%	70.11%
31	18.84%	71.61%	20.33%	69.89%
32	18.90%	71.66%	20.47%	69.79%
33	18.95%	71.54%	20.26%	70.02%
34	18.93%	71.59%	20.40%	69.84%
35	18.90%	71.66%	20.40%	69.84%
36	18.81%	71.74%	20.26%	70.02%
37	18.76%	71.80%	20.18%	70.15%
38	18.79%	71.80%	20.33%	70.06%
39	18.77%	71.83%	20.18%	70.15%
40	18.76%	71.87%	20.26%	70.11%
41	18.49%	72.41%	20.18%	70.48%
42	18.54%	72.35%	20.26%	70.19%
43	18.51%	72.40%	20.33%	70.06%
44	18.44%	72.41%	20.47%	69.79%