Churn Reduction

Shrikrishna Padalkar

November 2018

1 Introduction

Customer Churn is the loss of customers due to competition. Companies invest a substantial amount of resources in terms of both money and manpower to attract customers. It is more easy to lure an old customer than to get new customers.
Reducing the customer attrition will increase the profit making ability of the company. This report is all about understanding the reasons for Churn of Customers. We are going to elaborate on the methods used for predicting whether a Customer will churn out or remain with the company.

2 Understanding Churn Rate

When do we say that a particular customer has churned out? If the customer has not visited you within a specific period of time. This duration varies for different industrial verticals. For gaming industry if a customer did not show up within say 48 hours of visiting then it can be considered as a churn.

Churn Rate(or Attrition Rate) is the measure of customers moving out of a group over specific period of time. Mathematically churn rate can be formulated as follows:-

$$churn_rate = C/T \tag{1}$$

Where C represents number of customers churning out over a period of time, and T represents the total number of customers.

Life Time Value(LTV) Suppose a firm has 100 customers and the Company's expenditure is $1000/month. The monthly churn rate 10%. This implies that 10 customers are lost per month. The monthly revenue lost is $10000/month.

$$LTV = \frac{MonthlyProductionCost}{ChurnRate} \tag{2}$$

3 Check Class Imbalance

Class Imbalance occurs when the number of observations of one or more classes are very less as compared to the number of observations of other classes. Here, we have two classes Churned Employees and Not Churned Employees. There exist 2850 records of employees who remained with the organization and 483 employees churned out. Thus the difference is substantial.

Visual Representation
Following are a few techniques to deal with class imbalance:-

- **Under Sampling**:- In this technique we choose a subset of samples from the majority class in such a way that the number of samples are closer to the number of samples in the minority class.
  Here we have used Systematic Sampling as a choice of sampling technique. Consider a two class classification problem. Suppose there are N observations from the majority class and n from the minority class. We choose every $k^(th)$ observation from the majority class. Where k = N /n. This will result in the majority class having number of observations close to the minority class.
- **Over Sampling**:- In this technique we add dummy observations to the minority class in order to match the majority class.

4 Feature Selection

Feature selection is crucial part of any kind of analysis. We will find those features which have less correlation among themselves. Two separate plots one each for the two classes are shown below.

```
#Find the correlation between the numeric variables
def correlation(data, label):
    #Correlation
    numeric_variables = list(set(churned emp.columns) - set(
                                        categorical_variables))

    f, ax = plt.subplots(figsize=(7,5))
    data_corr = data[numeric_variables].corr()

    sbn.heatmap(data_corr,
                mask = np.zeros_like(data_corr, dtype=np.bool),
                cmap = sbn.diverging_palette(220, 20, as_cmap=True)
                                                    ,

                square = True,
                ax = ax)
```

As it's evident from the graphs that the amount of minutes consumed by a customer is proportional to the charges incurred. The features of interest are as follows:-

- total day calls
- total day charge
- total eve calls
- total eve charge
- total night calls
- total night charge

Analysis Churned Employees.png Analysis Churned Employees.png

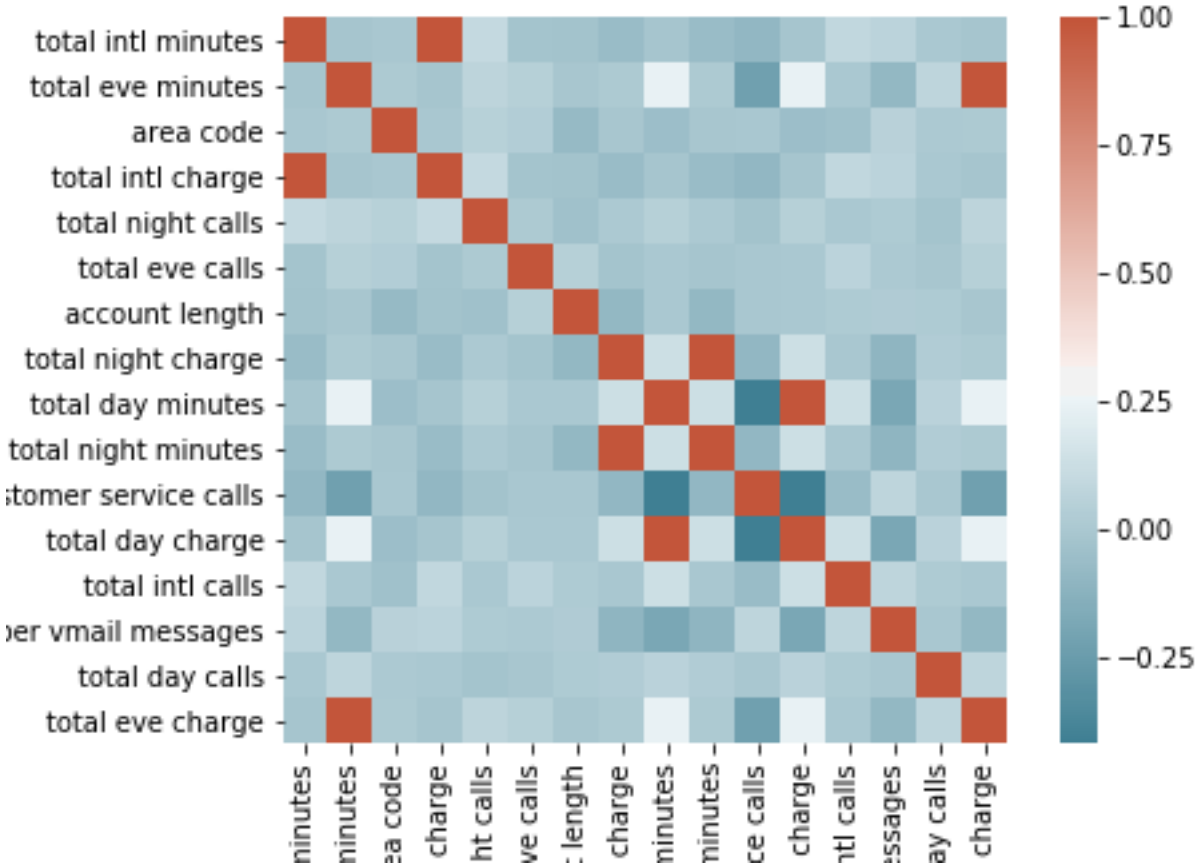


Figure 1: Correlation Analysis Churned Employees

Analysis Retained Employees.png Analysis Retained Employees.png

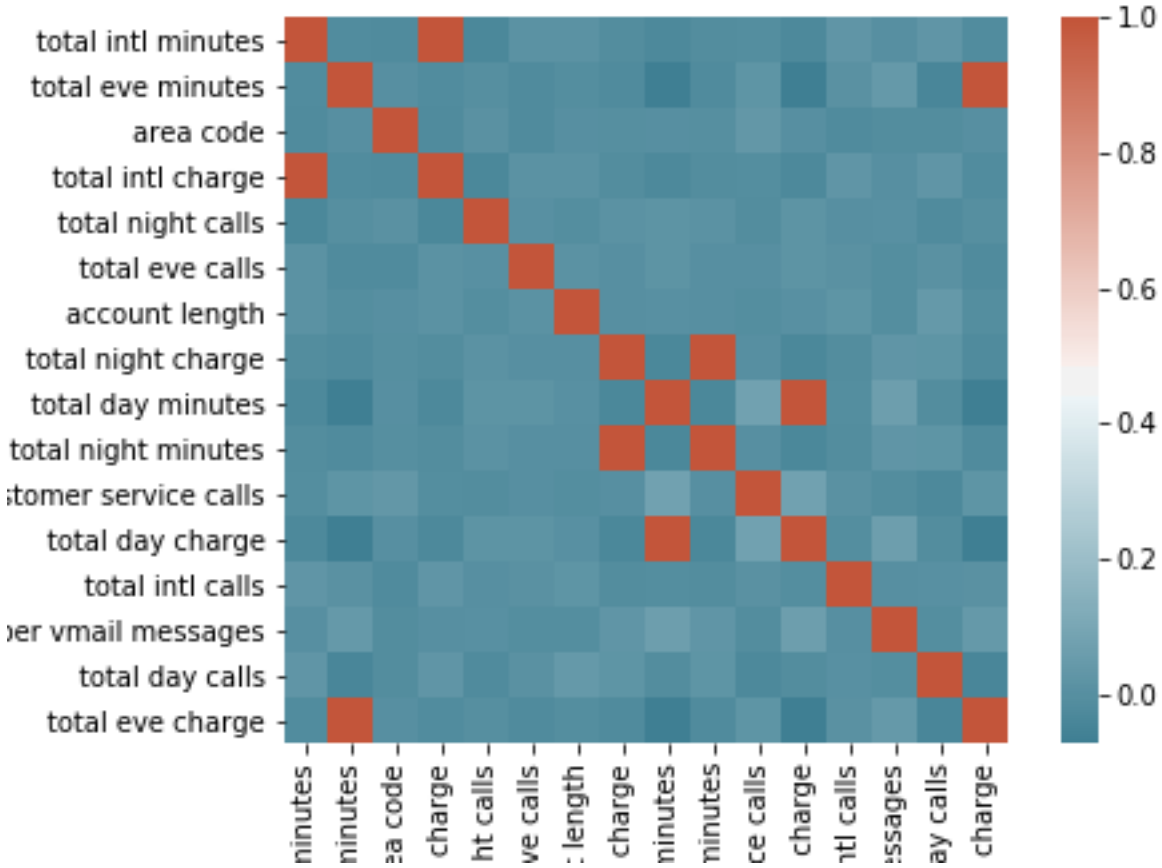


Figure 2: Correlation Analysis Retained Customers

Apart from these there are 3 categorical variables namely 'international plan', 'voice mail plan', 'Churn'. To understand the relationship between categorical variables we use **Chi Squared Test of Independence**. In **Chi Squared Test of Independence** the null hypothesis states that the two variables are independent. We then check whether we have strong evidence to reject the null hypothesis. Chi Squared test makes use of **p-value** to reject or to accept the claim.
If the **p-value** is less than 0.05 then we reject the null hypothesis stating that the two variables are dependent. We are checking the independence for

1. 'international plan' and 'Churn'
2. 'voice mail plan' and 'Churn'

```
for cat in categorical_variables[:-1]:
    print(cat)
    chi2, p, dof, ex = chi2_contingency(pd.crosstab(churn_combine['
                                        Churn'], churn_combine[cat]))
    print("{}\t{}\t{}\t{}\n" .format(chi2, p, dof, ex))
```

The results are as shown below:-

Variable	Chi SquaredStatistic	P-value	DOF	Expected Value
international plan	79.36	5.17e-19	1	[[392.19 82.80] [398.80 84.19]]
voice mail plan	23.83	1.04e-06	1	[[364.43 110.56] [370.56 112.43]]

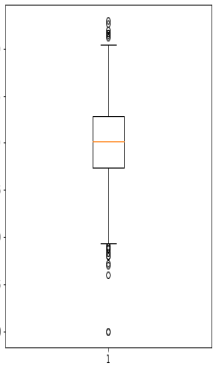
5 Outlier Analysis

Outliers are the extreme values in the data that tend to inflate the variance and increase the error. These are nothing but extreme values.
We use Tuckey's method for outlier analysis. Here is the code and it's result.

```
def plotting(variable):
    f, ax = plt.subplots(figsize=(6,6))
    plt.boxplot(churn_train[variable])
    plt.savefig(os.getcwd() + "/Plots/" + variable + ".png")

def outlier_analysis(variable):
    #Plotting the variable
    plotting(variable)

    #Set the lower and upper quartile
    data = []
    data = input().split(' ')
    lower, upper = np.percentile(churn_train[variable], [int(data[0
                                    ]), int(data[1])])

    #IQR
    IQR = (upper-lower)

    lower_outliers = churn_train.loc[churn_train[variable]<(lower -
                                     1.5*IQR), variable]
    upper_outliers = churn_train.loc[churn_train[variable]>(upper +
                                     1.5*IQR), variable]

    #Capping the outlier
    churn_train.loc[churn_train[variable]>(upper + 1.5*IQR),
                                    variable] = (upper + 1.5*IQR)
    churn_train.loc[churn_train[variable]<(lower - 1.5*IQR),
                                    variable] = (lower - 1.5*IQR)

    print("Lower:- {}\nUpper:- {}" .format(lower_outliers,
                                    upper_outliers))
```

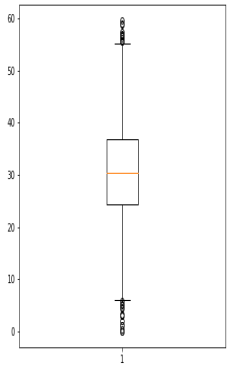
Observations:-

- The variables are uniformly distributed (not skewed)
- The means of the variables 'Day Calls', 'Night Calls' and 'Evening Calls' are closer.



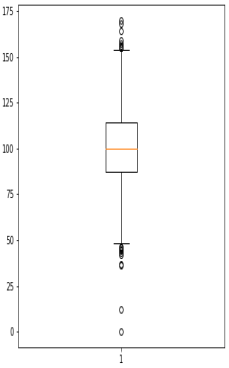
day calls.png day calls.png

(a) first



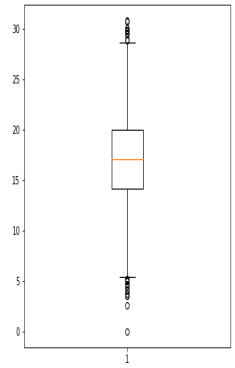
day charge.png day charge.png

(b) second



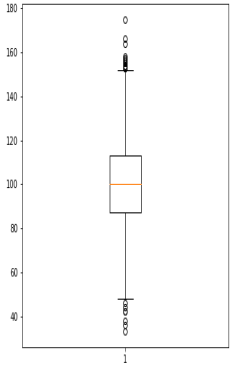
eve calls.png eve calls.png

(c) third



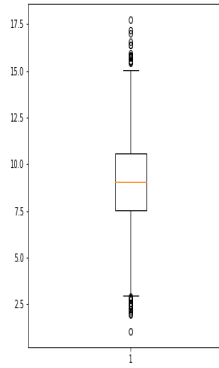
eve charge.png eve charge.png

(d) fourth



night calls.png night calls.png

(e) fifth



night charge.png night charge.png

(f) sixth

6 Modeling

We have applied various algorithms in order to predict the target class on the training data. The target is a binary variables as seen earlier. Thus we can use binary classification algorithmns such as logistic regression. Every algorithm has certain assumptions. Lets explore a few of them and compare the results.

```
class modeling():

    from sklearn.metrics import roc_curve, roc_auc_score
    from sklearn.metrics import confusion_matrix, f1_score

    def __init__(self, X_train, X_test, y_train, y_test, model,
                                        name_of_classifier):
        self.X_train = X_train
        self.X_test = X_test
        self.y_train = y_train
        self.y_test = y_test
        self.model = model
        self.clf_name = name_of_classifier

    def auc_roc(self):
        predicted_probability = self.model.predict_proba(self.
                                            X_test)

        fpr, tpr, _ = roc_curve(self.y_test,  predicted_probability
                                            [::,1])
        auc = roc_auc_score(self.y_test, predicted_probability[::,1
                                            ])
        plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
        plt.legend(loc=4)
        plt.xlabel("False Positive Rate")
        plt.ylabel("True Positive Rate")
        plt.savefig("AUC " + self.clf_name + ".png")
        plt.show()

    def evaluate(self, CM):
        accuracy = CM.diagonal().sum()/CM.sum()
        precision = CM[0][0]/(CM[0][0] + CM[0][1])
        recall = CM[0][0]/(CM[0][0] + CM[1][0])
        f_measure = (2*precision*recall)/(precision+recall)
        print("Accuracy:- {}\nPrecision:- {}\nRecall:- {}\nF-
                                            measure:- {}" .format(
                                            accuracy,precision,
                                            recall, f_measure))

    def prediction(self, f=1):
        #X_train, X_test, y_train, y_test = train_test_split(self.X
                                            , self.y, test_size=
                                            test_size/100,
                                            random_state=42)

        #Fit the model
        model = self.model.fit(self.X_train, self.y_train)

        #Predict on test set
        predicted_test = model.predict(self.X_test)
```

```
        #Results
        result_test = pd.DataFrame({'Actual':self.y_test.ravel(), '
                                        Predicted':predicted_test
                                        .ravel()}, columns=['
                                        Actual','Predicted'])
        result_test.head()

        #Save the model
        joblib.dump(model, os.getcwd() + "/Models/" + self.clf_name
                                         + ".pkl")

        #Print the AUC_ROC curve
        self.auc_roc()

        #Evaluate the model
        CM = confusion_matrix(self.y_test.ravel(), predicted_test.
                                        ravel(), labels=None,
                                        sample_weight=None)
        self.evaluate(CM)
        #F1 Score
        #print(f1_score(y_test.ravel(), predicted_test.ravel()))
```

6.1 Logistic Regression

Logistic Regression is a binary classification algorithm based on an activation function. The inputs to the activation function are observations and the results are classes(0 or 1). There are variety of activation functions viz, **Sigmoid**, **tanh** etc. Let's have a look at the equation of the Sigmoid Function.

$$g(x) = \frac{e^x}{1 + e^x} \tag{3}$$

Let's have a look at the assumptions of Logistic Regression

- The independent variables must not be correlated
- The independent variables are linearly related to the log odds
- Include meaningful variables
- The dependent variables should be binary
- Sample size must be large

Logistic Regression.png Logistic Regression.png

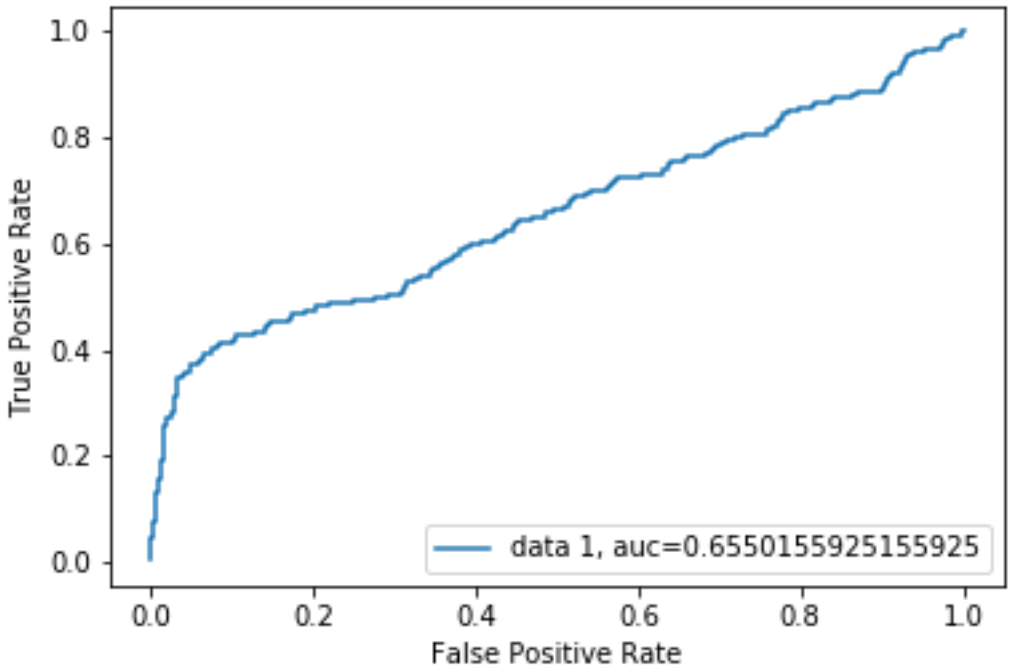


Figure 3: Logistic Regression

```
#Calculate the log odds of the target variable
#log(P/(1-P))
#P -- Probability of Churn
#1-P -- Probility of no Churn
P = len(churn_combine.loc[churn_combine['Churn']==1, 'Churn'])/len(
                                    churn_combine.loc[churn_combine['
                                    Churn']])
print("%.5f\t%.5f\n" %(P,(1-P)))
#Corrleation between Log Odds and Independent Variables
np.log(P/(1-P))
```

Results:

Metric	Values
Accuracy	0.57
Precision	0.57
Recall	0.90
F-measure	0.70
Predicted Churn Rate	0.49

	Actual Churn	Actual Not Churned
Predicted Churn	825	618
Predicted Not Churn	86	138

Naive Bayes Model.png Naive Bayes Model.png

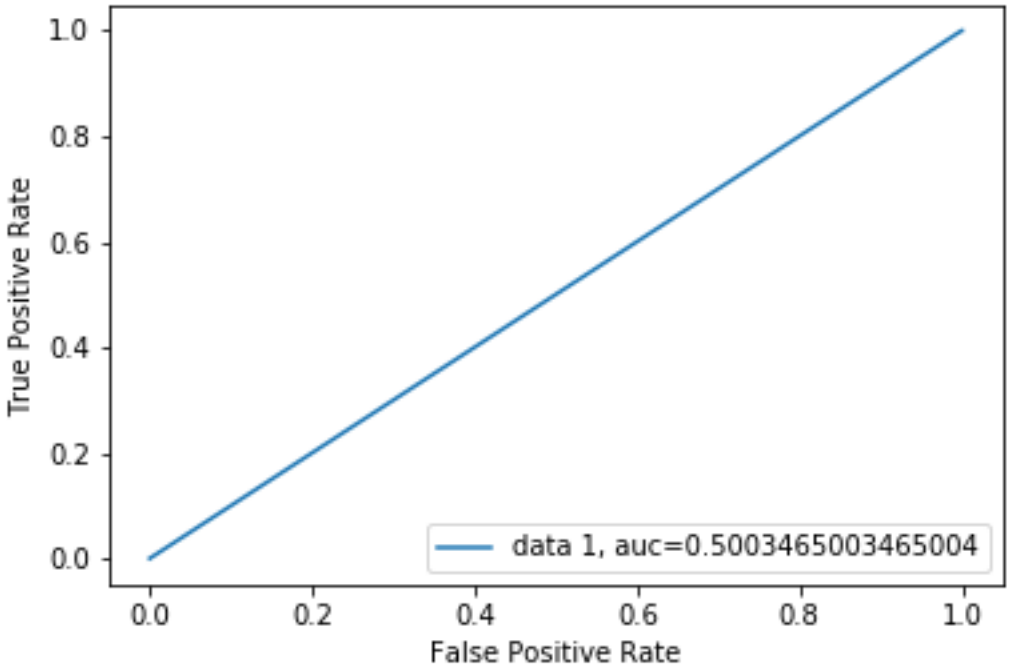


Figure 4: Naive Bayes Classification

6.2 Naive Bayes Classification

Naive Bayes Classifier is a probabilistic Classifier which makes use of conditional probability to predict the target. **Results:**

Metric	**Values**
Accuracy	0.13
Precision	0.0006
Recall	1.0
F-measure	0.0013
Predicted Churn Rate	0.0005

	Actual Churn	Actual Not Churned
Predicted Churn	1	1442
Predicted Not Churn	0	224

6.3 Support Vector Machines

Simple Support Vector Machines are basically designed to classify linearly separable data. The objective is to maximize the distance between the two classes(if the problem is Binary Classification). In case of multi-class classification problem **One Versus Rest(OVR)** approach is used.
However, if the data is non linearly separable then various transformations are applied on the data in order to make it linearly separable. These transformations usually send the data into higher dimensional space. This is known as **Kernel Trick**.
In our case the data is not linearly separable. We have used the default RBF kernel to train the SVC. Below are the results of the prediction

Results RBF Kernel:

Metric	Values
Accuracy	0.15
Precision	0.03
Recall	0.87
F-measure	0.05
Predicted Churn Rate	0.02

	Actual Churn	Actual Not Churned
Predicted Churn	44	1399
Predicted Not Churn	5	219

Support Vector Machine Model.png Support Vector Machine Model.png

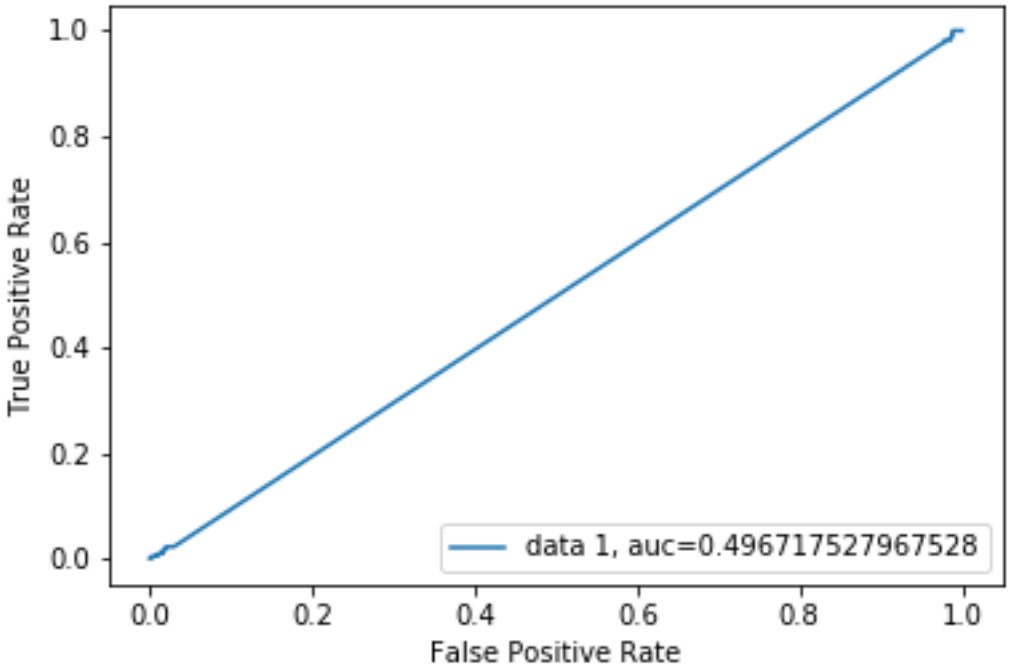


Figure 5: Support Vector Machine

6.4 Random Forest Classifier

Random Forest Classifier is an ensemble of Decision Tree Classifier. Individual Decision Trees are usually less accurate hence referred to as weak learners. But the combination of multiple Decision Trees have a propensity to provide good accuracy.

We have tried with 10(default), 20, and 80 Trees. Below are the results for each one of these.

Results:

Metric	**10 Trees**	**20 Trees**	**80 Trees**
Accuracy	0.70	0.68	0.69
Precision	0.72	0.70	0.70
Recall	0.90	0.91	0.91
F-measure	0.80	0.79	0.79
Predicted Churn Rate	0.62	0.60	0.61

- **Confusion Matrix(Random Forest with 10 Trees)**

	Actual Churn	Actual Not Churned
Predicted Churn	44	1399
Predicted Not Churn	5	219

- **Confusion Matrix(Random Forest with 20 Trees)**

	Actual Churn	Actual Not Churned
Predicted Churn	1013	430
Predicted Not Churn	91	133

- **Confusion Matrix(Random Forest with 80 Trees)**

	Actual Churn	Actual Not Churned
Predicted Churn	1024	419
Predicted Not Churn	95	129

Random Forest Model 80 Trees.png Random Forest Model 80 Trees.png

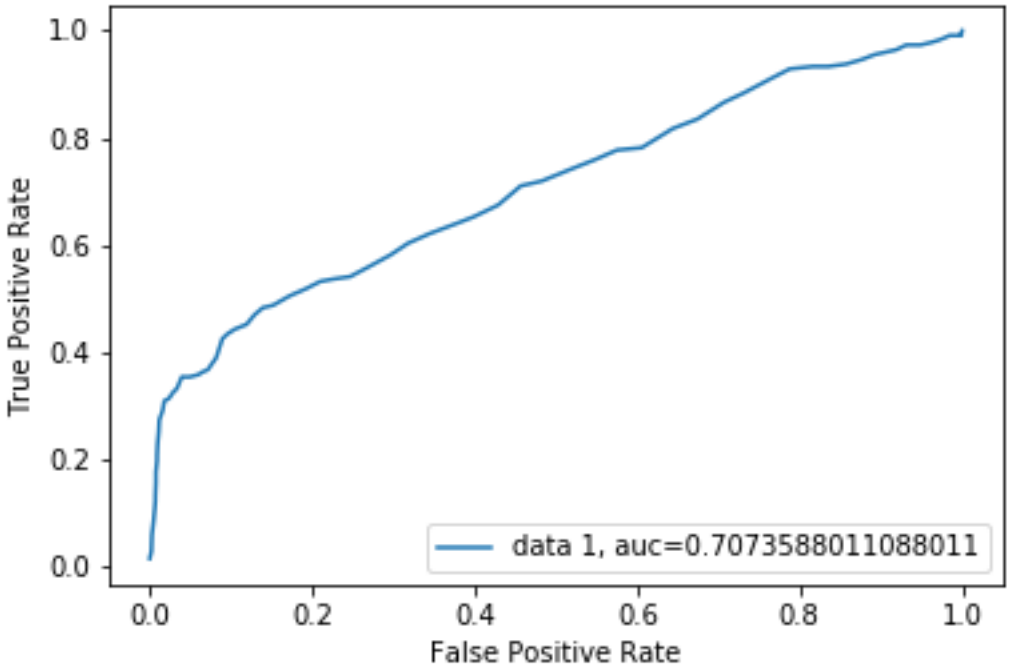


Figure 6: Random Forest Model 80 Trees

7 Conclusion

We are going to evaluate performances of various models based on Accuracy, F-Measure and Area Under the Curve(AUC). Since the objective is to predict the **Churn Rate** we will go with the model that has maximum AUC and least False Negative Rate. False Negatives are the cases where the algorithm predicts that a particular customer will not churn out, but actually the customer churns out. This will be a loss to the company. On the other hand False Positives are less severe as compared to False Negatives.

Algorithm	Predicted Churn Rate	AUC	FNR
Logistic Regression	0.70	0.65	0.09
Naive Bayes	0.0005	0.50	0
Support Vector Classifier	0.02	0.49	0.10
Random Forest(10 Trees)	0.62	0.68	0.09
Random Forest(20 Trees)	0.60	0.69	0.08
Random Forest(80 Trees)	0.61	0.70	0.08