Title: Telecom Enterprise Customer Churn Prediction

IE7275 Data Mining in Engineering

Sayam Khatri (002877613)

Table of Contents

1. Title: Telecom Enterprise Customer Churn Prediction
2. Abstract
3. Objective
4. Introduction/ Proposal
5. Dataset Selection and Preprocessing
6. Exploratory Data Analysis (EDA) and Feature Selection
7. Model Implementation and Baseline Evaluation
8. Hyperparameter Tuning
9. Model Evaluation and Comparative Analysis
10. Conclusion and Recommendations

Abstract

Examining customer characteristics as features and issuing early alerts for potential churn using supervised machine learning algorithms can enable companies to tailor marketing strategies and personalized services, launch discount, schemes, and various offerings to specific group of people leading to significant cost savings and preventing losses.

With Data Mining & ML, I conducted various preprocessing tasks such as data cleaning, undersampling, oversampling, and standardization on a dataset comprising 900,000 telecom customer profiles and historical behaviors. I then employed K-Nearest Neighbors (KNN), Random Forest (RF), and XGBoost models, alongside other machine learning approaches like Naive Bayes, Neural Networks, and Logistic Regression, to analyze customer churn data.

Results indicate that the first three models exhibit superior performance in terms of recall rate, precision rate, F1 score, and other metrics, with KNN and XGBoost demonstrating the best performance. Specifically, the recall rates for NN, RF, and XGBoost on positive samples are 84%, 83%, and 88%, respectively, while the precision rates are 84%, 82%, and 89%, and the F1 scores are 77%, 82%, and 88%. Compared to the benchmarks, the KNN & XGBoost model outperforms with recall, precision, and F1 scores reaching 88%, 90%, and 90%, respectively.

These predictive insights on customer churn offer valuable data support for telecom companies to implement targeted retention strategies for at-risk customers, thereby mitigating churn rates.

Objective

This project aims to provide hands-on experience in applying various supervised learning algorithms to a selected dataset, performing feature selection, tuning hyperparameters, and evaluating model performance using appropriate metrics. The project should compare at least three different supervised learning algorithms to understand their strengths, weaknesses, and applicability to specific types of data and problems.

- **Select the dataset of your choice for supervised problems**: Model training and analysis utilized real user data sourced from a specific operator. The dataset comprises personal attributes and behavioral data of various users over a three-month period, totaling 900,000 entries. Each record has 35 feature variables such as user ID, contract status, online duration, call status, and traffic usage. The final column is the target variable which indicates whether the user has churned. A value of 0 denotes a negative sample representing users who have not churned, while a value of 1 signifies a positive sample indicating users who have churned.

- **Understand and apply different supervised learning algorithms**: Exploratory Data Analysis was done to explore and better understand the data in hand. Supervised Algorithms like KNN, Neural Networks, Naïve Bayes, Random Forest, XGBoost, Logistic Regression were used

- **Perform feature selection to improve model performance:** Extensive feature selection was done. 35 variables were reduced to 14 features through feature engineering and considering domain knowledge. True Outliers were kept in baseline model evaluation but were removed further to improve model performance. RFE was conducted and the feature adding up the least importance was removed.

- **Tune hyperparameters to optimize models:** Hyper parameter tuning was performed using Grid Search CV with cross- validation=5. The best parameters are feed to the later models and performance is compared further.

- **Evaluate and compare model performance using various metrics:** Various metrics like precision, recall, accuracy, f1-score for both positive samples as well as negative samples are considered to assess model performance.

Introduction/Proposal

In the current scenario, where the number of domestic mobile communication users is closely equal to China's total population and the telecommunications market is evenly divided among the three major operators, the remaining customer base is increasingly light. So, the disparity in maintenance costs between new and existing customers is becoming more noticeable. **Referring to the American Marketing Association's Customer Satisfaction Handbook, acquiring a new customer costs five times more than retaining an existing one.** Thus, the decisions of communication operators should shift from a 'product-centric' approach to one that is 'customer-centric'.

Telecom operators has a huge user base, making it impossible to feed individual demands of each customer. However, if the company can accurately forecast customer churn, it stands to maintain major human, material, and financial resources. Additionally, it can implement targeted retention strategies, enhance customer loyalty, and leverage the sustainable profitability and low maintenance costs associated with existing customers. This, in turn, elevates the core competitiveness of the enterprise.

With the rising reputation of machine learning, ML models are increasingly preferred for addressing customer churn within the telecom industry.

Analyses are conducted on customer personal information and historical behavioral datasets from telecom operators. Five classic learning models—KNN, RF, XGBoost, NN, and Naïve Bayes are chosen for this purpose. The coding is executed using Python language, in the Jupyter Notebook development tool. The assessment of the models is based on their predictive performance, measured through metrics such as average precision rate, recall rate, and F1 score. Furthermore, the prediction outcomes are visually represented via ROC curves. **The learning model demonstrating the highest recall rate should be selected for early warning systems regarding customer churn.**

Dataset Selection and Preprocessing

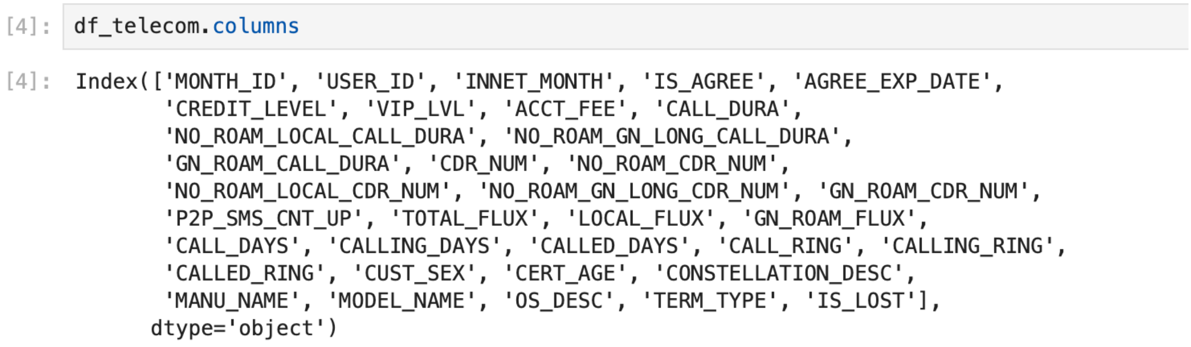
Data sources

https://figshare.com/articles/dataset/S1_Data_-/24292936

In this project, real world user data obtained from a specific operator in China is utilized for model training and analysis purposes. I got the data from the above link. There is an article published about the same data using different models and feature space. The original dataset from the operator has personal attributes and behavioral data of various users over a span of three months, totaling 900,000 entries. Each data record comprises 35 feature variables, including user ID, contract status, online duration, call status, and traffic usage, among others. Notably, the final column is the dependent variable, indicating whether the user has churned or not.

Data Cleaning and Preprocessing

Columns:



```
[4]: df_telecom.columns
```

```
[4]: Index(['MONTH_ID', 'USER_ID', 'INNET_MONTH', 'IS_AGREE', 'AGREE_EXP_DATE',
            'CREDIT_LEVEL', 'VIP_LVL', 'ACCT_FEE', 'CALL_DURA',
            'NO_ROAM_LOCAL_CALL_DURA', 'NO_ROAM_GN_LONG_CALL_DURA',
            'GN_ROAM_CALL_DURA', 'CDR_NUM', 'NO_ROAM_CDR_NUM',
            'NO_ROAM_LOCAL_CDR_NUM', 'NO_ROAM_GN_LONG_CDR_NUM', 'GN_ROAM_CDR_NUM',
            'P2P_SMS_CNT_UP', 'TOTAL_FLUX', 'LOCAL_FLUX', 'GN_ROAM_FLUX',
            'CALL_DAYS', 'CALLING_DAYS', 'CALLED_DAYS', 'CALL_RING', 'CALLING_RING',
            'CALLED_RING', 'CUST_SEX', 'CERT_AGE', 'CONSTELLATION_DESC',
            'MANU_NAME', 'MODEL_NAME', 'OS_DESC', 'TERM_TYPE', 'IS_LOST'],
           dtype='object')
```

1. **MONTH_ID:** The month id in which customer joined the company
2. **USER_ID:** Unique User ID of each customer
3. **INNET_MONTH:** Number of months customer has committed to the company
4. **IS_AGREE:** Does the customer agree on some Special Terms & Conditions
5. **AGREE_EXP_DATE:** Date of Agreeing
6. **CREDIT_LEVEL:** Certain Credit Level of the Customer
7. **VIP_LVL:** Certain VIP Level of the Customer according to the company
8. **ACC_FEE:** Fee Paid by the Customer
9. **CALL_DURA:** Duration of Calls
10. **NO_ROAM_LOCAL_CALL_DURA', 'NO_ROAM_GN_LONG_CALL_DURA', 'GN_ROAM_CALL_DURA':** Duration of Calls Roaming/Nonroaming/ Distance wise
11. **CDR_NUM:** Number of call Details
12. **NO_ROAM_CDR_NUM', 'NO_ROAM_LOCAL_CDR_NUM', 'NO_ROAM_GN_LONG_CDR_NUM', 'GN_ROAM_CDR_NUM':** Specific call details Roaming wise
13. **P2P_SMS_CNT_UP:** Peer to Peer SMS Sent
14. **TOTAL_FLUX:** Total Flux used by customer
15. **LOCAL_FLUX', 'GN_ROAM_FLUX:** Flux used in roaming/long Distance
16. **CALL_DAYS, CALLING_DAYS, CALLED_DAYS:** Number of days call/Called
17. **CALL_RING', 'CALLING_RING', 'CALLED_RING:** Number of days specific Rings
18. **CUST_SEX:** Gender of the customer
19. **CERT_AGE:** Age of the Customer according to some id they provided
20. **CONSTELLATION_DESC:** Zodiac Sign of Customer
21. **MANU_NAME:** Manuscript name of Customer
22. **MODEL_NAME:** Phone Model
23. **OS_DESC:** Operating System of the phone
24. **TERM_TYPE:** Type of term
25. **IS_LOST:** Is the customer lost? 1- Lost, 0 - Not

Missing values:

AGREE_EXP_DATE: **440665**
VIP_LVL: **311283**
CUST_SEX: **34287**
CERT_AGE: **34971**
OS_DESC: **38194**
IS_LOST: **599691**

Our Target variable has 599691 missing values. Rows with target variable as missing are dropped and kept as a separate data to predict whether the customer will churn or not based on model training through the available data

It is not clearly specified in the data source destination about AGREE_EXP_DATE, so the column has several missing values as well, thus is dropped during model training.

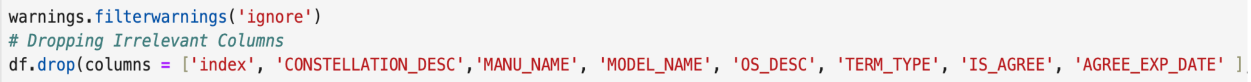
OS_DESC is dropped as most of the customers are using the same Operating system

CUST_SEX and CERT AGE have 3% of missing values in the data set. Having good quantity of data, we can drop this as well.

VIP_LVL can be an important column to determine the customer churn, but it has three categories 99, 2, 3 which does not show any relationship with IS_LOST. Thus, this column is dropped from the dataset as part of feature engineering

There are 321 duplicate values found in the data, that were removed.

MONTH_ID of each customer is same, thus dropped from our Data.



```
warnings.filterwarnings('ignore')
# Dropping Irrelevant Columns
df.drop(columns = ['index', 'CONSTELLATION_DESC','MANU_NAME', 'MODEL_NAME', 'OS_DESC', 'TERM_TYPE', 'IS_AGREE', 'AGREE_EXP_DATE' ],
```

Some Irrelevant columns like customer zodiac sign, manuscript name, model name and OS name, Term type which is same for every customer are dropped. Proper details are not specified for IS Agree and Agree exp date, thus these columns are removed from the data.

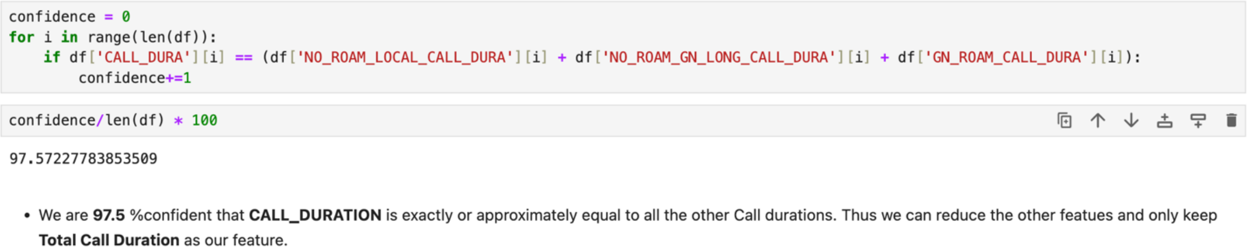
IS LOST and Cert Age has float dtypes, which are converted to int dtype for proper interpretation.

Exploratory Data Analysis (EDA) and Feature Selection

```
CREDIT_LEVEL
67   99386
66   98918
65   86967
0      17
```

No Certain evidence of how Credit level plays a role in customer churn because all the data points are equally divided in these categories, plus the column categories is ambiguous. We will drop this column.

We will Keep **Total_flux** instead of LOCAL_FLUX (FLUX of Data used Locally) and GN_ROAM_FLUX (FLUX of data used in roaming) as a part of feature reduction



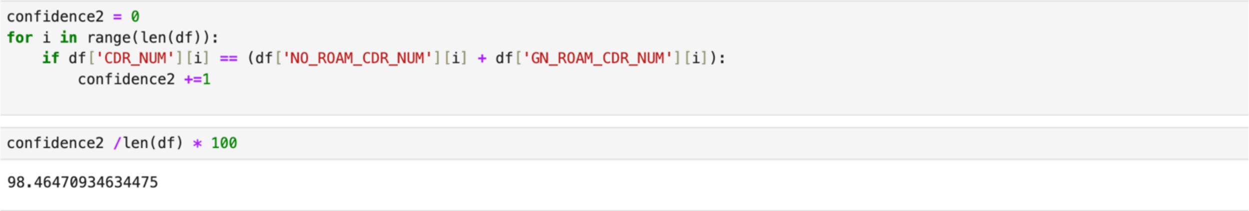
```
confidence = 0
for i in range(len(df)):
    if df['CALL_DURA'][i] == (df['NO_ROAM_LOCAL_CALL_DURA'][i] + df['NO_ROAM_GN_LONG_CALL_DURA'][i] + df['GN_ROAM_CALL_DURA'][i]):
        confidence+=1
```

```
confidence/len(df) * 100
```

```
97.57227783853509
```

- We are **97.5** %confident that **CALL_DURATION** is exactly or approximately equal to all the other Call durations. Thus we can reduce the other featues and only keep **Total Call Duration** as our feature.

CALL_DURA is the sum of **NO_ROAM_LOCAL_CALL_DURA', 'NO_ROAM_GN_LONG_CALL_DURA', 'GN_ROAM_CALL_DURA', thus** we can only keep CALL_DURA column as total call duration of a customer and drop other columns.



```
confidence2 = 0
for i in range(len(df)):
    if df['CDR_NUM'][i] == (df['NO_ROAM_CDR_NUM'][i] + df['GN_ROAM_CDR_NUM'][i]):
        confidence2 +=1
```

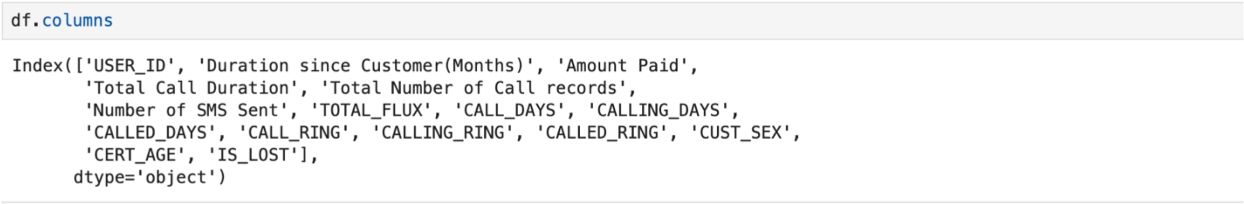
```
confidence2 /len(df) * 100
```

```
98.46470934634475
```

Similarly, we can check for **CDR_NUM** (Number of call details records), which is sum of (NO_ROAM_CDR_NUM, GN_ROAM_CDR_NUM). We are 98.4 % confident that CDR_NUM is exactly or approximately equal to **NO_ROAM_CDR_NUM + GN_ROAM_CDR_NUM**. Thus, we can reduce the other features and only keep total call details as our feature.

Now, we will clean the names of the columns to better understand the data.

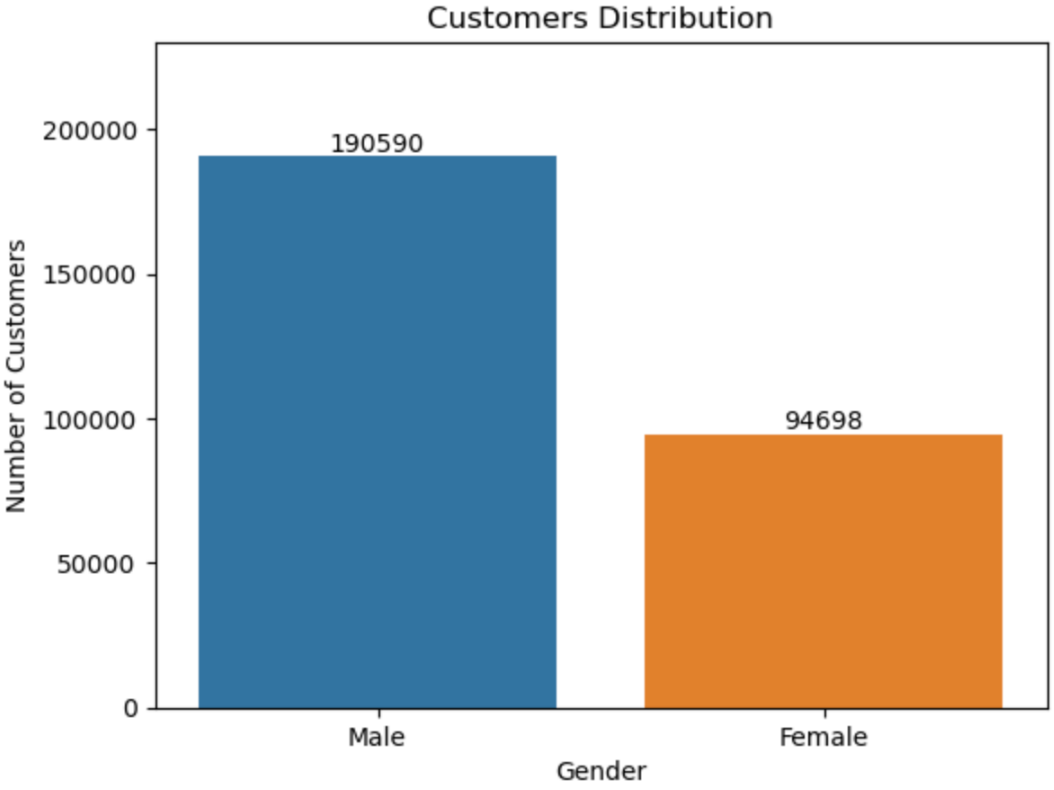
Shape of data frame: (283140, 16)



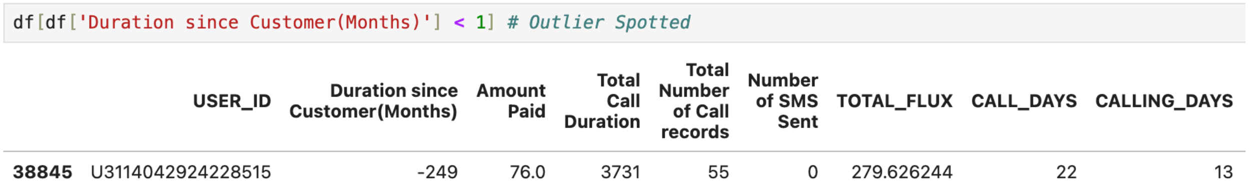
```
df.columns
```

```
Index(['USER_ID', 'Duration since Customer(Months)', 'Amount Paid',
       'Total Call Duration', 'Total Number of Call records',
       'Number of SMS Sent', 'TOTAL_FLUX', 'CALL_DAYS', 'CALLING_DAYS',
       'CALLED_DAYS', 'CALL_RING', 'CALLING_RING', 'CALLED_RING', 'CUST_SEX',
       'CERT_AGE', 'IS_LOST'],
      dtype='object')
```

We are now left with 16 relevant columns, instead of 35 and thus can carry out with EDA now.

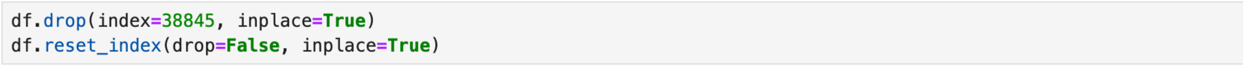


The data contains 19,0590 number of males and 94,698 number of females.



```
df[df['Duration since Customer(Months)'] < 1] # Outlier Spotted
```

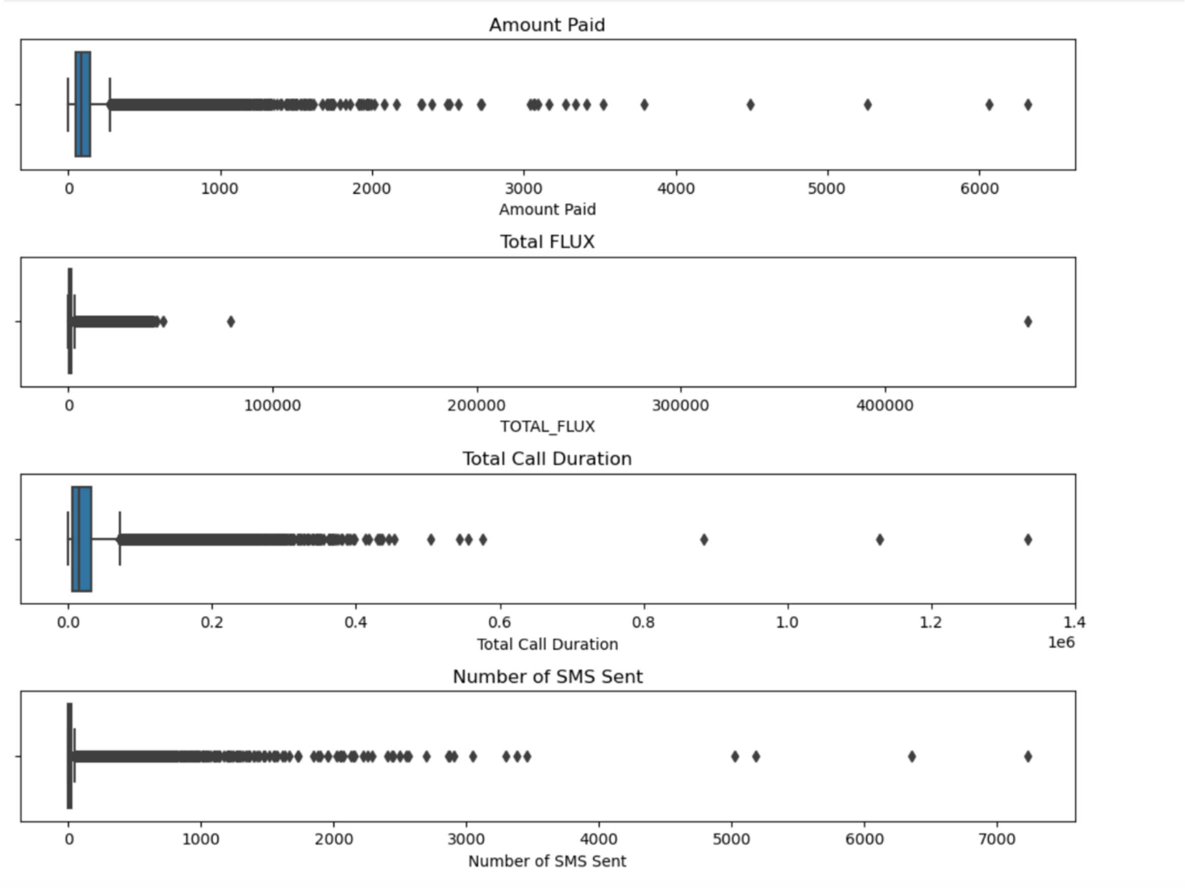
	USER_ID	Duration since Customer(Months)	Amount Paid	Total Call Duration	Total Number of Call records	Number of SMS Sent	TOTAL_FLUX	CALL_DAYS	CALLING_DAYS
38845	U3114042924228515	-249	76.0	3731	55	0	279.626244	22	13



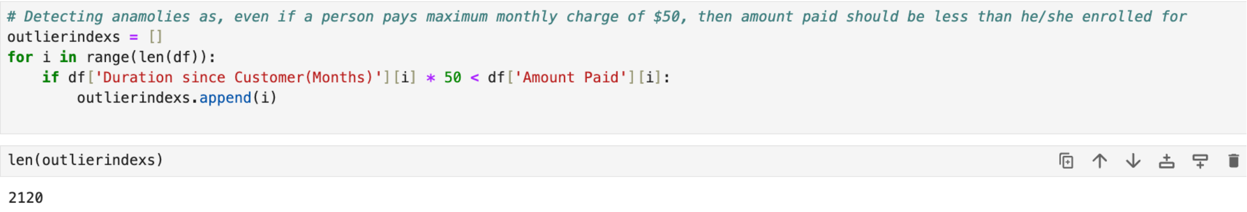
```
df.drop(index=38845, inplace=True)
df.reset_index(drop=False, inplace=True)
```

An outlier was spotted, as the duration of Customers cannot be negative, thus it is removed.

Identifying more Outliers,



Many outliers are detected, must be dealt with keeping domain knowledge in mind

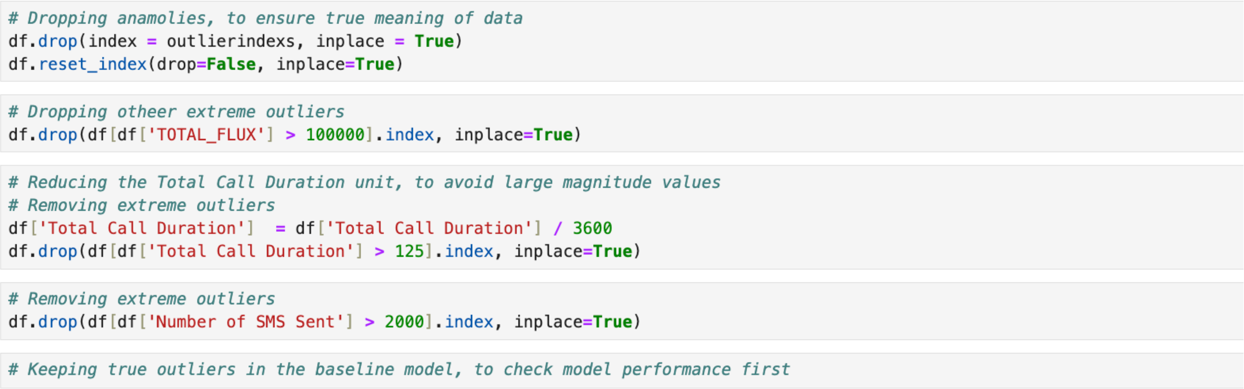


```
# Detecting anamolies as, even if a person pays maximum monthly charge of $50, then amount paid should be less than he/she enrolled for
outlierindexs = []
for i in range(len(df)):
    if df['Duration since Customer(Months)'][i] * 50 < df['Amount Paid'][i]:
        outlierindexs.append(i)
```

```
len(outlierindexs)
```

```
2120
```

Since, the company offers maximum monthly plan of $50, Anomalies are checked, and 2120 anomalies are found where Amount paid is greater than 50 * Duration of Enrollment. These anomalies are removed. True outliers are kept in the baseline evaluation and may be removed further to check model performance.



```
# Dropping anamolies, to ensure true meaning of data
df.drop(index = outlierindexs, inplace = True)
df.reset_index(drop=False, inplace=True)
```

```
# Dropping otheer extreme outliers
df.drop(df[df['TOTAL_FLUX'] > 100000].index, inplace=True)
```

```
# Reducing the Total Call Duration unit, to avoid large magnitude values
# Removing extreme outliers
df['Total Call Duration']  = df['Total Call Duration'] / 3600
df.drop(df[df['Total Call Duration'] > 125].index, inplace=True)
```

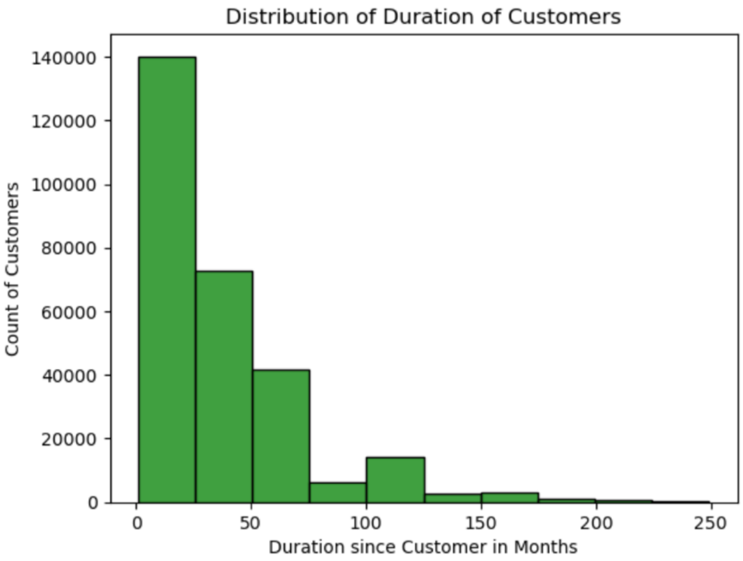
```
# Removing extreme outliers
df.drop(df[df['Number of SMS Sent'] > 2000].index, inplace=True)
```

```
# Keeping true outliers in the baseline model, to check model performance first
```

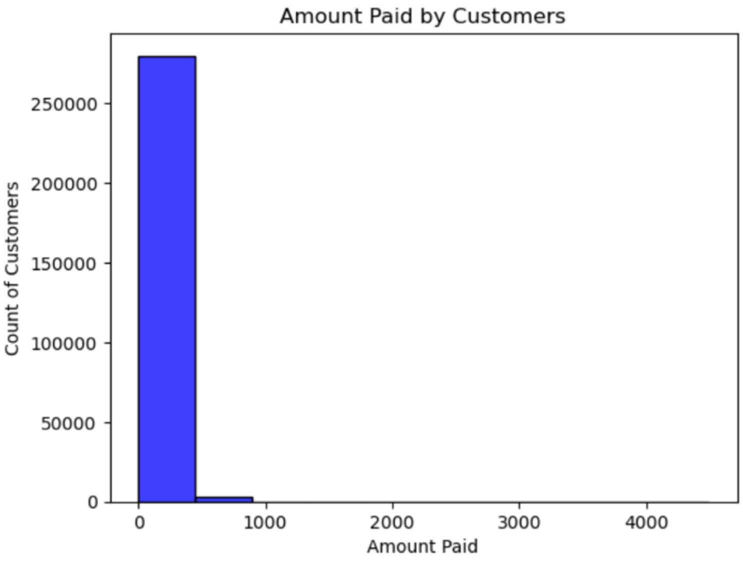
Extreme outliers are removed, but some outliers are kept considering the importance of features in determining the usage of the customer of the telecom service, which will contribute to customer churn prediction.

Total call duration is divided by 3600, to convert into hours and normalize it for better model performance.

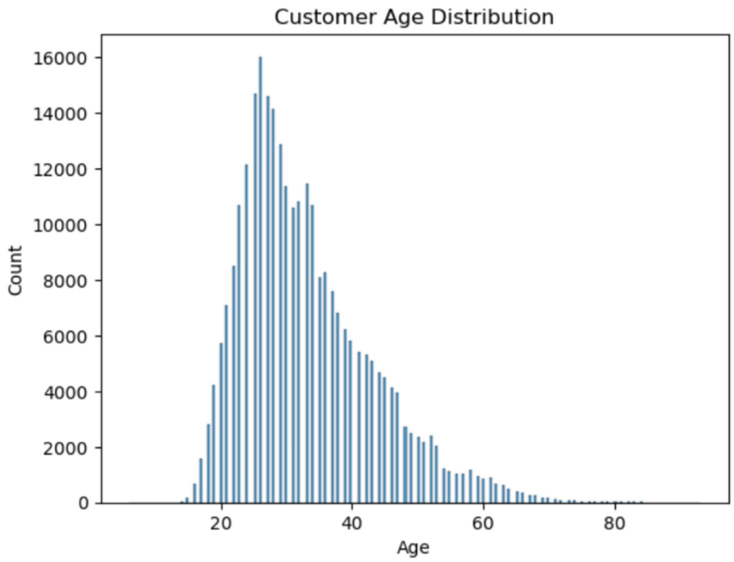
We can visually check the distribution of various columns to check data skewness.



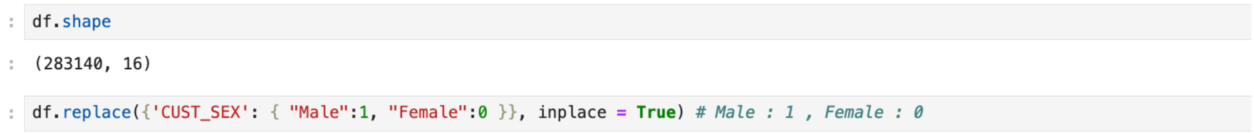
Majority of the customers are enrolled in 0–50-month period.



$0-$500 are spent by majority of the customers; some true outliers are kept in the data considering the domain knowledge.



Most of the customers are between age 20-40, and distribution looks a little right skewed.



```
: df.shape

: (283140, 16)

: df.replace({'CUST_SEX': { "Male":1, "Female":0 }}, inplace = True) # Male : 1 , Female : 0
```

Encoding customer sex column to 1 & 0.

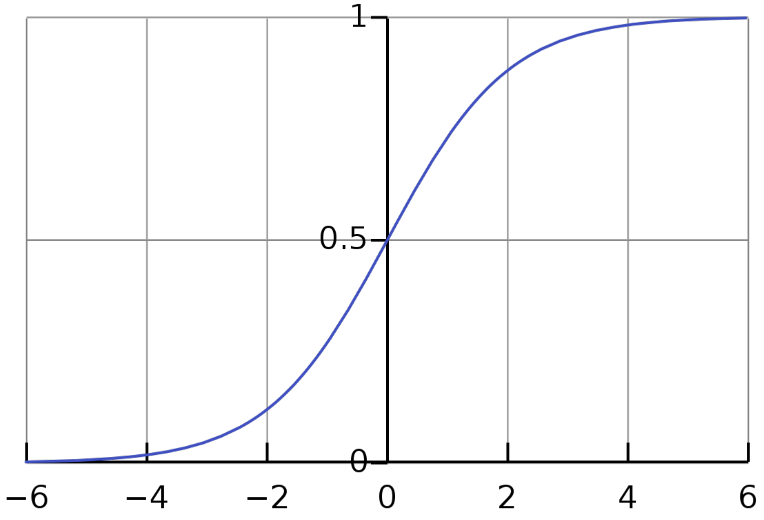
Model Implementation and Baseline Evaluation

Baseline Evaluation

Thought the modeling process, the data is split using random split of test size as 0.2, using **X_train,X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2 , random_state = 42)**

Logistic Regression

Logistic regression is a model for binary classification tasks, where the target variable has only two possible outcomes, i.e. 0 and 1. In logistic regression, the relationship between the independent variables (features) and the binary dependent variable is modeled using the logistic function. This function maps any real-valued input into the range [0, 1], making it suitable for modeling probabilities.



Our data is split using train test split and fitted in Logistic Regression Model. The instance of LR model is created using sklearn package.

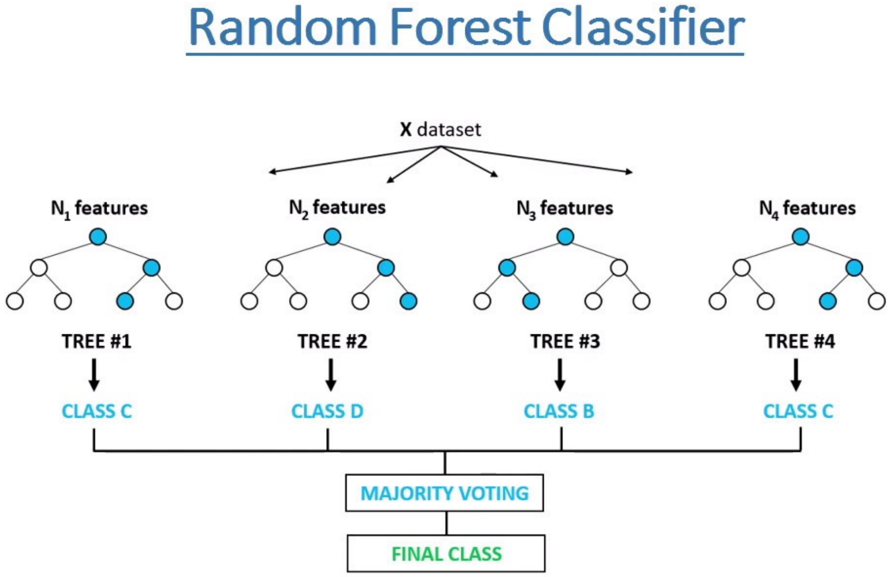
```
'0':
{'precision': 0.9998721531240298, 'recall': 0.9668850779746031, 'f1-score': 0.98310198071},
'1':
{'precision': 0.0, 'recall': 0.0, 'f1-score': 0.0, 'accuracy': 0.9667655576746486}
```

Our model can classify negative samples well, but cannot identify positive samples, which was our main goal here.

The reason for this is class imbalance, which we will deal with ahead. Ensemble methods are known to deal good with class imbalance, we will also check baseline performance using Random Forest Classifier.

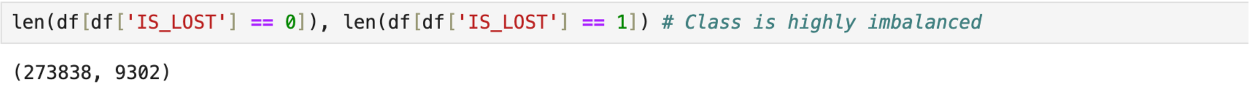
Random Forest Classifier

Random Forest Classifier is a versatile and powerful ensemble learning method used for both classification and regression tasks. It belongs to the family of decision tree-based algorithms and is known for its high accuracy, robustness, and ability to handle large datasets with high dimensionality.



```
{'0':
 {'precision': 0.9964385513122569, 'recall': 0.9748418682771682, 'f1-score': 0.98552190680912
94, 'support': 55966.0},
 '1': {'precision': 0.24906666666666666, 'recall': 0.7054380664652568, 'f1-score': 0.3681513598
7386677, 'support': 662.0},
'accuracy': 0.9716924489651763,
```

Random forest classifier can perform on class imbalance as well, with a moderate recall rate. In general, it handles class imbalance well. But still, the precision value is very low, which cannot be ignored. Thus, we will handle all the issues using Undersampling the majority class and over sampling the minority class.



```
len(df[df['IS_LOST'] == 0]), len(df[df['IS_LOST'] == 1]) # Class is highly imbalanced
```

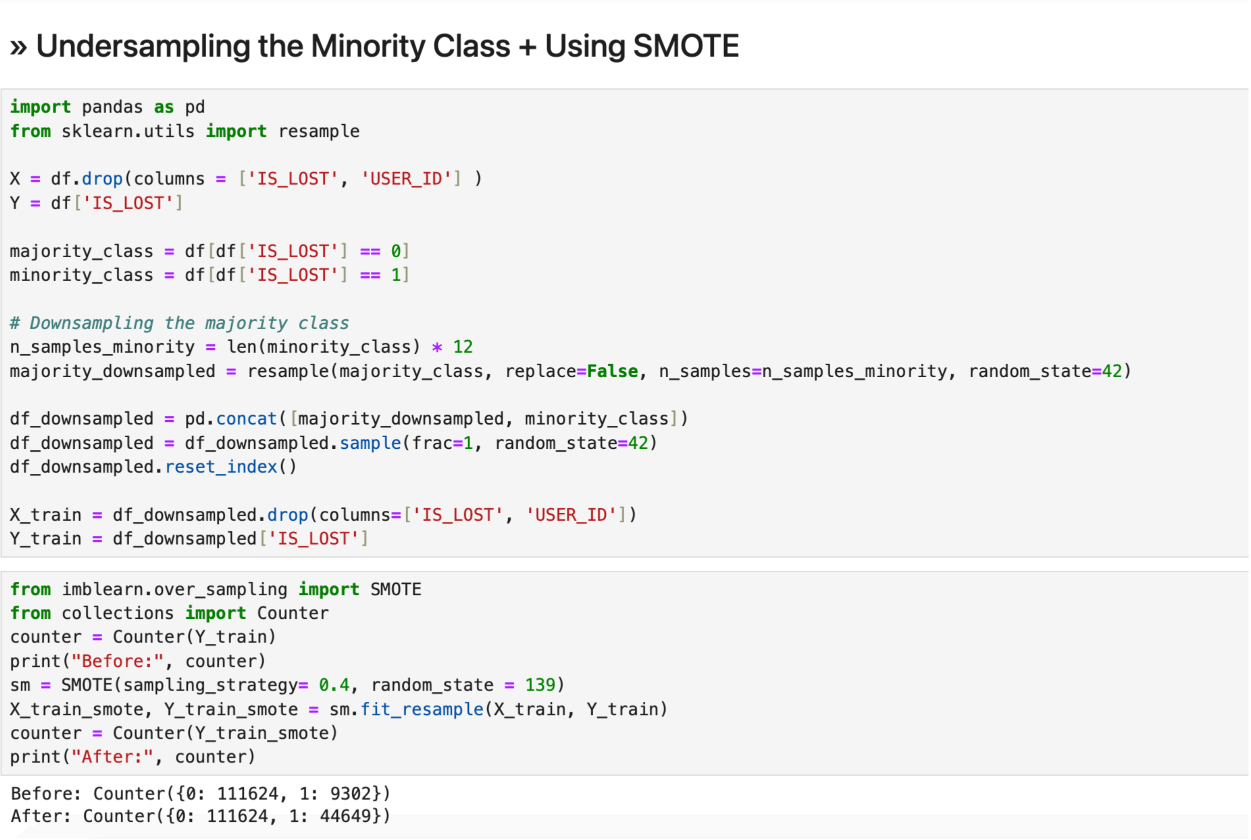
```
(273838, 9302)
```

There is huge difference between minority class (class 1) and majority class (class 0). We will handle class imbalance using undersampling the majority class and oversampling the minority class using SMOTE.

SMOTE: Synthetic Minority Oversampling Technique

SMOTE is an oversampling technique where the synthetic samples are generated for the minority class. This algorithm helps to overcome the overfitting problem posed by random oversampling. It focuses on the feature space to generate new instances with the help of interpolation between the positive instances that lie together.

Thus, we approach by down sampling our majority class data to 12 times the minority class and then using SMOTE for oversampling the majority class.



» Undersampling the Minority Class + Using SMOTE

```
import pandas as pd
from sklearn.utils import resample

X = df.drop(columns = ['IS_LOST', 'USER_ID'] )
Y = df['IS_LOST']

majority_class = df[df['IS_LOST'] == 0]
minority_class = df[df['IS_LOST'] == 1]

# Downsampling the majority class
n_samples_minority = len(minority_class) * 12
majority_downsampled = resample(majority_class, replace=False, n_samples=n_samples_minority, random_state=42)

df_downsampled = pd.concat([majority_downsampled, minority_class])
df_downsampled = df_downsampled.sample(frac=1, random_state=42)
df_downsampled.reset_index()

X_train = df_downsampled.drop(columns=['IS_LOST', 'USER_ID'])
Y_train = df_downsampled['IS_LOST']
```

```
from imblearn.over_sampling import SMOTE
from collections import Counter
counter = Counter(Y_train)
print("Before:", counter)
sm = SMOTE(sampling_strategy= 0.4, random_state = 139)
X_train_smote, Y_train_smote = sm.fit_resample(X_train, Y_train)
counter = Counter(Y_train_smote)
print("After:", counter)
```

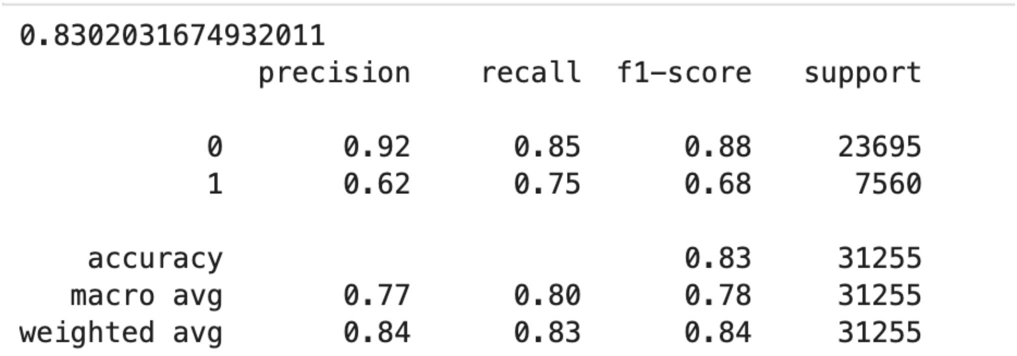
```
Before: Counter({0: 111624, 1: 9302})
After: Counter({0: 111624, 1: 44649})
```

Using these techniques, results in the data that is available for training as,
Before: Counter ({0: 111624, 1: 9302})
After: Counter ({0: 111624, 1: 44649})

We used a sampling strategy as 0.4 in this case, which gives us the ratio between the minority class and the majority class as 0.4.

Undersampling the data had to be performed to improve the performance of the model, given the data at hand.

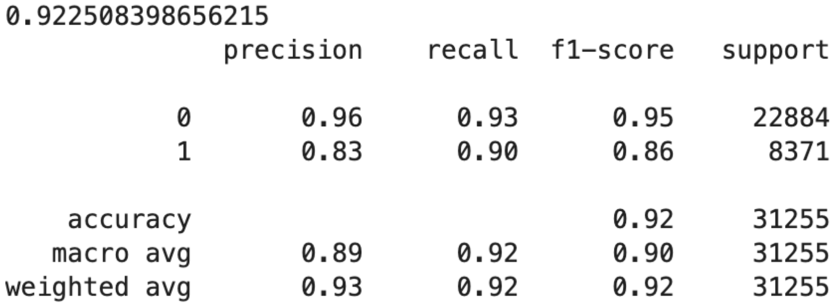
1. Logistic Regression



```
0.8302031674932011
              precision    recall  f1-score   support

           0       0.92      0.85      0.88     23695
           1       0.62      0.75      0.68      7560

    accuracy                           0.83     31255
   macro avg       0.77      0.80      0.78     31255
weighted avg       0.84      0.83      0.84     31255
```

We can see an improvement in performance, after using our down sampling and over sampling techniques. Still the model requires improvement due to the reason that can be multicollinearity between 2-3 columns in the dataset, and we will consider implementing other models for the classification.

2. Random Forest Classifier

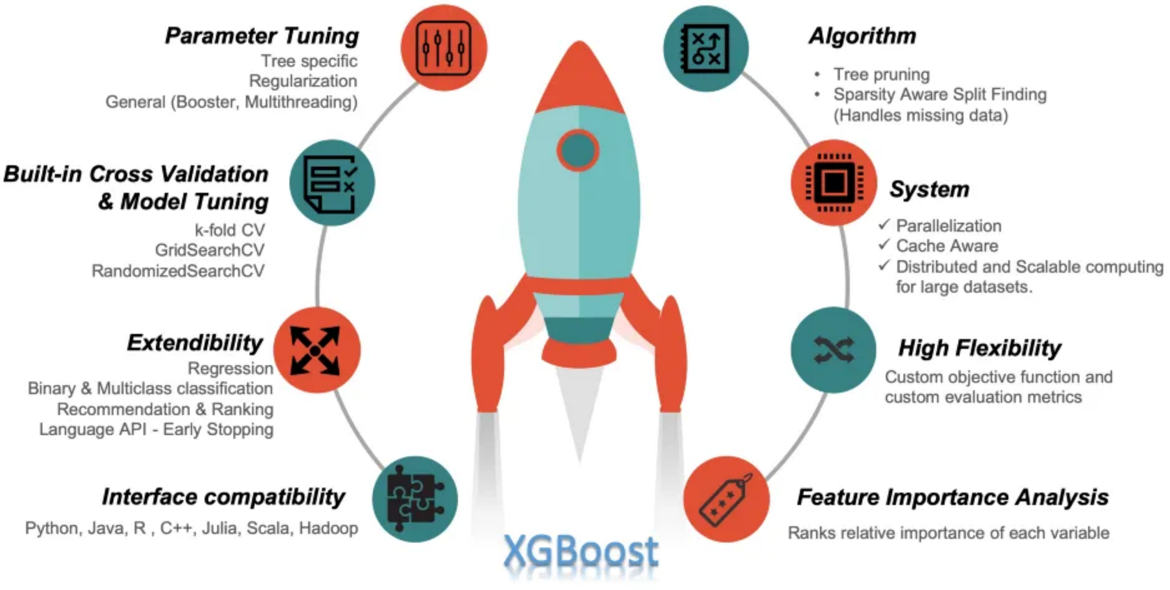


```
0.922508398656215
              precision    recall  f1-score   support

           0       0.96      0.93      0.95     22884
           1       0.83      0.90      0.86      8371

    accuracy                           0.92     31255
   macro avg       0.89      0.92      0.90     31255
weighted avg       0.93      0.92      0.92     31255
```

Random Forest Classifier can generalize the data well, for both the negative samples as well as the positive samples. The improvement in performance is due to undersampling as well as SMOTE.

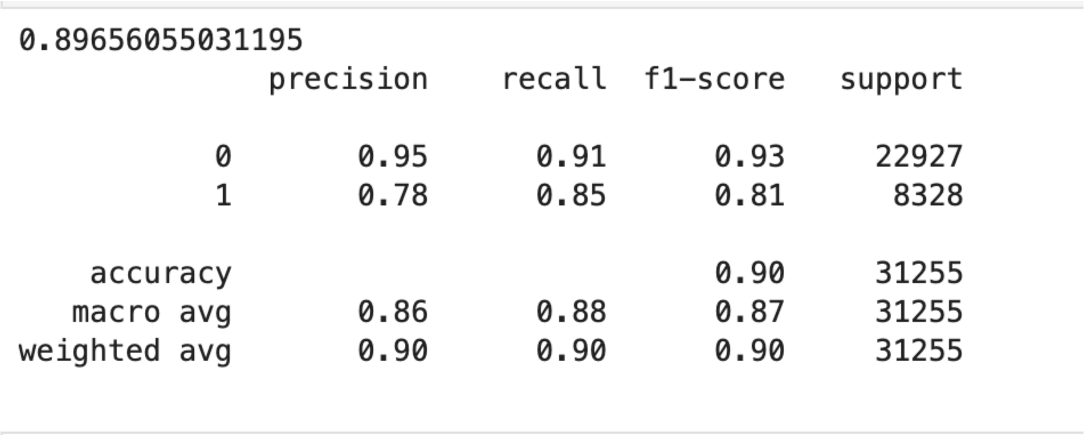
3. XGBoost Model

XGBoost is an algorithm that belongs to the ensemble learning category, specifically the gradient boosting framework. It utilizes decision trees as base learners and employs regularization methods to enhance model performance. It is known for its computational efficiency, feature importance analysis, and handling of missing values.



Parameter Tuning
Tree specific
Regularization
General (Booster, Multithreading)
Built-in Cross Validation & Model Tuning
k-fold CV
GridSearchCV
RandomizedSearchCV
Extendibility
Regression
Binary & Multiclass classification
Recommendation & Ranking
Language API - Early Stopping
Interface compatibility
Python, Java, R , C++, Julia, Scala, Hadoop
XGBoost
Algorithm
• Tree pruning
• Sparsity Aware Split Finding (Handles missing data)
System
✓ Parallelization
✓ Cache Aware
✓ Distributed and Scalable computing for large datasets.
High Flexibility
Custom objective function and custom evaluation metrics
Feature Importance Analysis
Ranks relative importance of each variable

XGBoost Model perform moderately good on the data. Base XGB Model is used to fit the data, but we will implement this model again, with tuned hyperparameters, which will increase the performance.

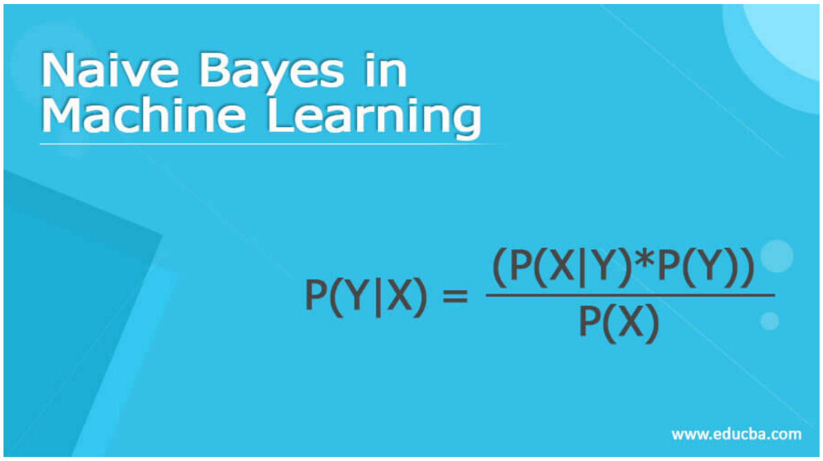


```
0.89656055031195
              precision    recall  f1-score   support

           0       0.95      0.91      0.93     22927
           1       0.78      0.85      0.81      8328

    accuracy                           0.90     31255
   macro avg       0.86      0.88      0.87     31255
weighted avg       0.90      0.90      0.90     31255
```

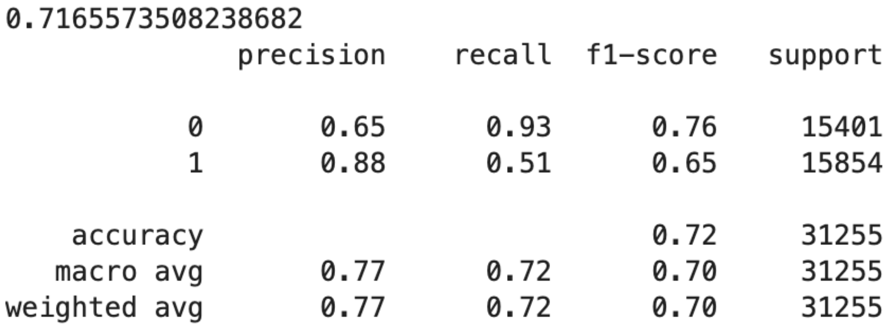
4. Naïve Bayes Classifier

Naive Bayes classifier is a probabilistic machine learning model based on Bayes' theorem. It assumes independence between features and calculates the probability of a given input belonging to a particular class. It's widely used in text classification, spam filtering, and recommendation systems.

We will implement our data using this model and try to measure its metrics.


Naive Bayes in Machine Learning
P(Y|X) = (P(X|Y)*P(Y)) / P(X)
www.educba.com

Naïve Bayes is not able to perform good on this data, with very less recall value for the positive sample. Even after undersampling the data and then oversampling it. It also records a low number of f1 score (0.76) and precision (0.65) for the negative class as well, where the other models do not struggle.

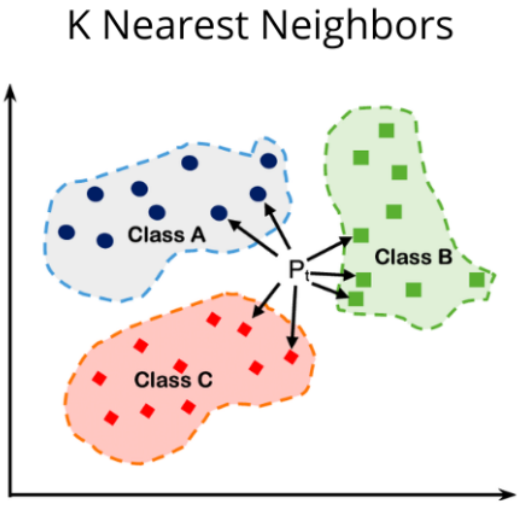


```
0.7165573508238682
              precision    recall  f1-score   support

           0       0.65      0.93      0.76     15401
           1       0.88      0.51      0.65     15854

    accuracy                           0.72     31255
   macro avg       0.77      0.72      0.70     31255
weighted avg       0.77      0.72      0.70     31255
```

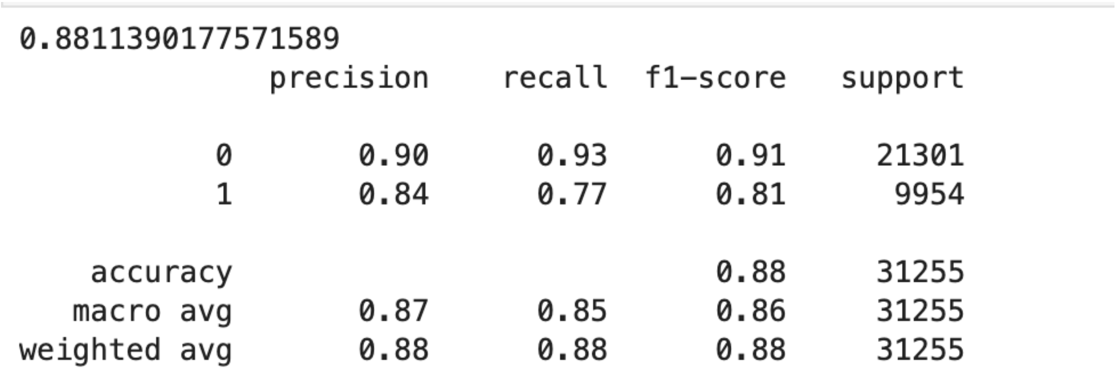
5. KNN

The K-Nearest Neighbors (KNN) algorithm is a popular machine learning technique used for classification and regression tasks. It relies on the idea that similar data points tend to have similar labels or values.

During the training phase, the KNN algorithm stores the entire training dataset as a reference. When making predictions, it calculates the distance between the input data point and all the training examples, using a chosen distance metric such as Euclidean distance.



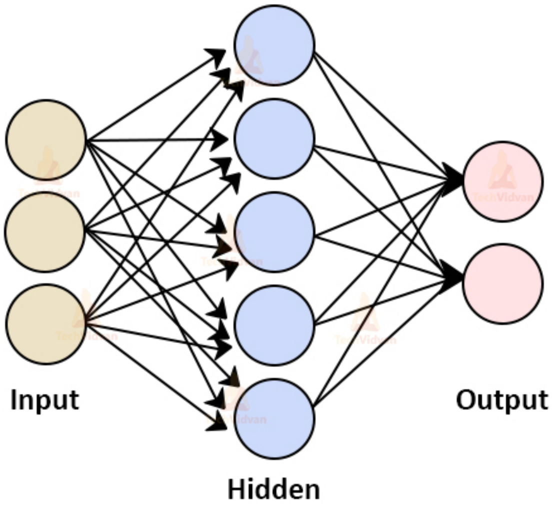
KNN, being a distance-based algorithm, our features must be scaled before the training phase to avoid biased results. We use Standard Scaler to scale the data before training into our model. KNN performs well on the data. We will tune the hyperparameter to check further performance.



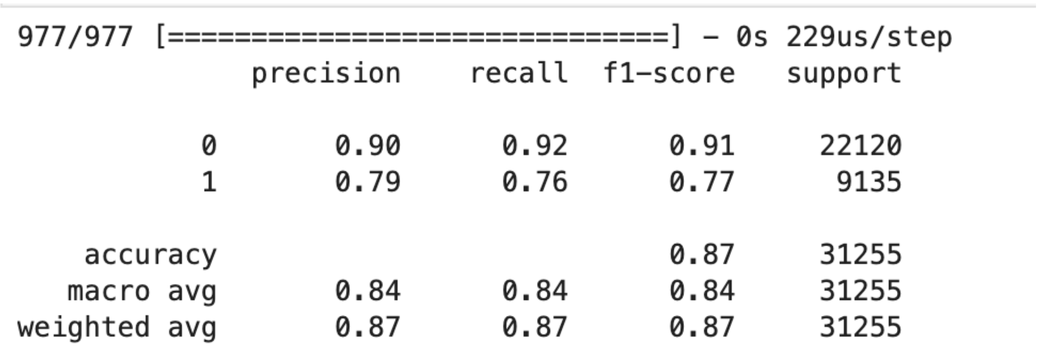
```
0.8811390177571589
              precision    recall  f1-score   support

           0       0.90      0.93      0.91     21301
           1       0.84      0.77      0.81      9954

    accuracy                           0.88     31255
   macro avg       0.87      0.85      0.86     31255
weighted avg       0.88      0.88      0.88     31255
```

6. Neural Networks

A Feed Forward Neural Network is an artificial Neural Network in which the nodes are connected circularly. The feed-forward model is the basic type of neural network because the input is only processed in one direction. The data always flows in one direction and never backwards/opposite. We are using feed forward neural network in our implementation and will manually hyper parameter tune in the model improvement phase.



Data entering the NN, should be scaled, thus we are using the same scaled data that we used in KNN training. NN performs well with the data but suffers a low recall than other models for positive samples.



```
977/977 [==============================] - 0s 229us/step
              precision    recall  f1-score   support

           0       0.90      0.92      0.91     22120
           1       0.79      0.76      0.77      9135

    accuracy                           0.87     31255
   macro avg       0.84      0.84      0.84     31255
weighted avg       0.87      0.87      0.87     31255
```

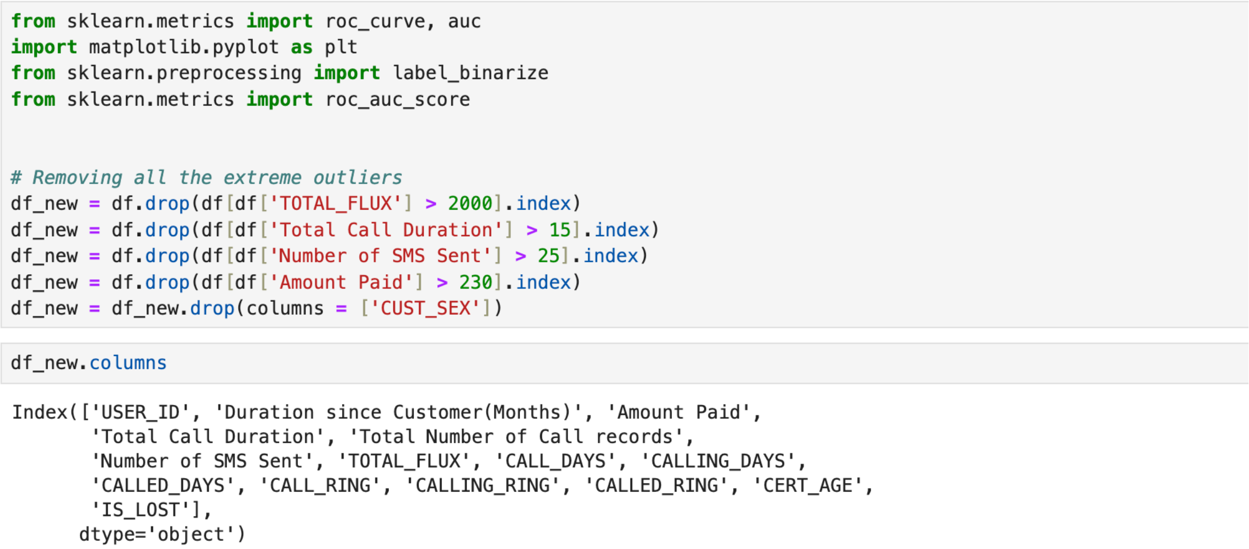
Hyperparameter Tuning

During hyperparameter tuning, I aim to refine the models by eliminating all the outliers. This tuning process is facilitated through techniques like Grid Search CV.

GridSearch CV

Grid Search thoroughly explores various combinations of specified hyperparameters and their corresponding values to assess model performance. However, this process is time-consuming and resource-intensive, mostly as the number of hyperparameters increases.
In the setting of GridSearchCV, which combines Grid Search with cross-validation, the model training process becomes iterative. Cross-validation partitions the training data into subsets, iteratively using each subset as both training and testing data. This iterative approach contributes to the overall time consumption during hyperparameter optimization.

Consequently, the combination of Grid Search and cross-validation significantly extends the time required to identify the best hyperparameters for the model.

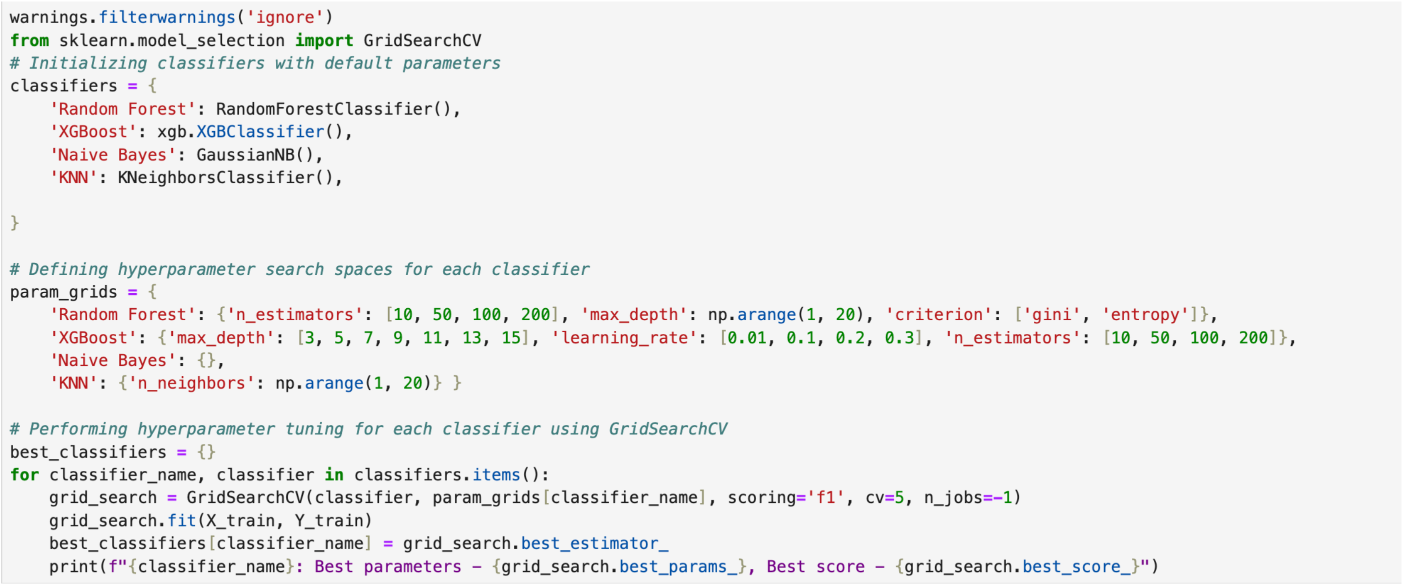


```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_auc_score


# Removing all the extreme outliers
df_new = df.drop(df[df['TOTAL_FLUX'] > 2000].index)
df_new = df.drop(df[df['Total Call Duration'] > 15].index)
df_new = df.drop(df[df['Number of SMS Sent'] > 25].index)
df_new = df.drop(df[df['Amount Paid'] > 230].index)
df_new = df_new.drop(columns = ['CUST_SEX'])
```

```
df_new.columns
```

```
Index(['USER_ID', 'Duration since Customer(Months)', 'Amount Paid',
       'Total Call Duration', 'Total Number of Call records',
       'Number of SMS Sent', 'TOTAL_FLUX', 'CALL_DAYS', 'CALLING_DAYS',
       'CALLED_DAYS', 'CALL_RING', 'CALLING_RING', 'CALLED_RING', 'CERT_AGE',
       'IS_LOST'],
      dtype='object')
```

Implementing GridSearch CV,



```
warnings.filterwarnings('ignore')
from sklearn.model_selection import GridSearchCV
# Initializing classifiers with default parameters
classifiers = {
    'Random Forest': RandomForestClassifier(),
    'XGBoost': xgb.XGBClassifier(),
    'Naive Bayes': GaussianNB(),
    'KNN': KNeighborsClassifier(),

}

# Defining hyperparameter search spaces for each classifier
param_grids = {
    'Random Forest': {'n_estimators': [10, 50, 100, 200], 'max_depth': np.arange(1, 20), 'criterion': ['gini', 'entropy']},
    'XGBoost': {'max_depth': [3, 5, 7, 9, 11, 13, 15], 'learning_rate': [0.01, 0.1, 0.2, 0.3], 'n_estimators': [10, 50, 100, 200]},
    'Naive Bayes': {},
    'KNN': {'n_neighbors': np.arange(1, 20)} }

# Performing hyperparameter tuning for each classifier using GridSearchCV
best_classifiers = {}
for classifier_name, classifier in classifiers.items():
    grid_search = GridSearchCV(classifier, param_grids[classifier_name], scoring='f1', cv=5, n_jobs=-1)
    grid_search.fit(X_train, Y_train)
    best_classifiers[classifier_name] = grid_search.best_estimator_
    print(f"{classifier_name}: Best parameters - {grid_search.best_params_}, Best score - {grid_search.best_score_}")
```

Due to environment limitations, I am trying limited number of combinations of hyperparameters, and will manually tune neural networks.

Random Forest Classifier: The performance of random forest classifier remains somewhat constant after the tuning. There is slight improvement in predicting negative samples. Since, I specified f1 score as the scoring metric in grid search, it tries to improve the average f1 score of the previous model.

XGBoost Model: XGBoost model performs well, after hyperparameter tuning. The precision, recall and the f1 score for both positive and negative samples increased.

KNN: KNN performance also increased after hyper parameter tuning. It recorded high recall value for both negative and positive samples.

Naïve Bayes: We cannot hyper parameter tune it, but its performance remained somewhat same after outlier reduction.

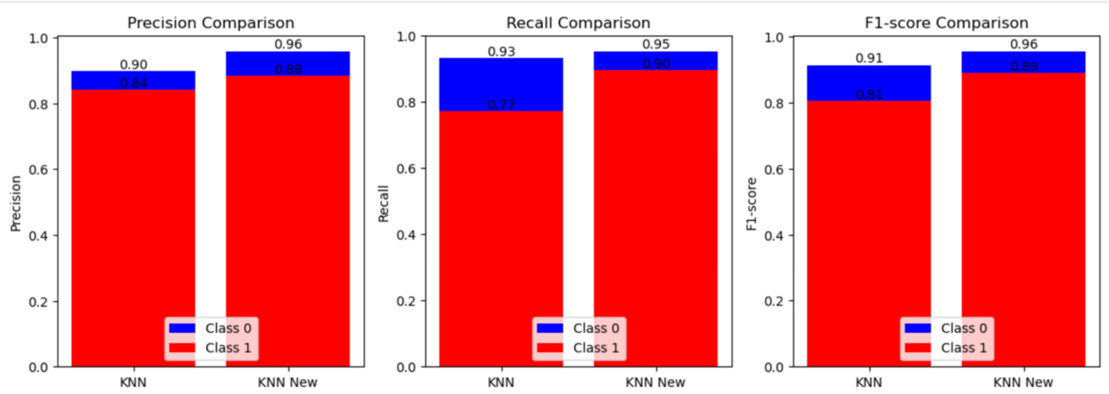
Neural Networks: After manual hyperparameter tuning of neural networks, the performance remains almost similar as I am not updating the weighs because it is a feed-forward neural network only. Number of hidden layers nodes, epochs and activation functions were changed.

Model Evaluation and Comparative Analysis

KNN

Best parameters - {'n_neighbors': 2}, Best score - 0.8974234789399809

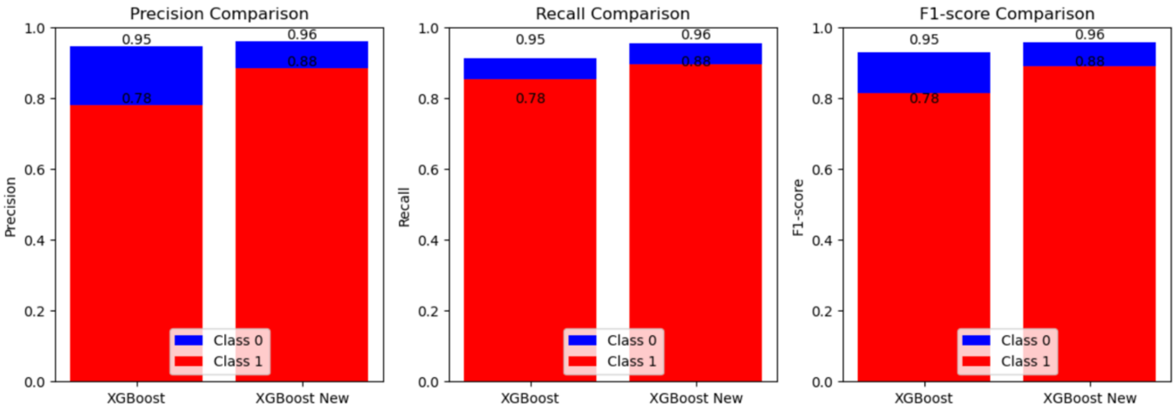
After the hyper parameter tuning of number of neighbors, KNN performance increased for both positive and negative samples. Specifically for positive samples the performance metrics increased by approximately 8-9% reference.



XGBoost

Best parameters - {'learning_rate': 0.3, 'max_depth': 15, 'n_estimators': 200}, Best score - 0.8898209360971056

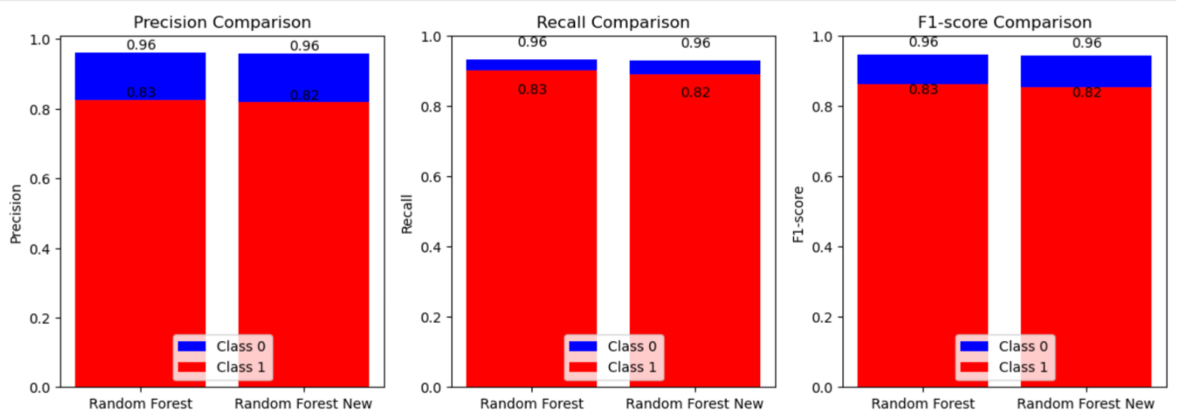
XGBoost performed well after hyperparameter tuning. It showed and increase in recall almost by a factor of 13% than previous.



Random Forest Classifier

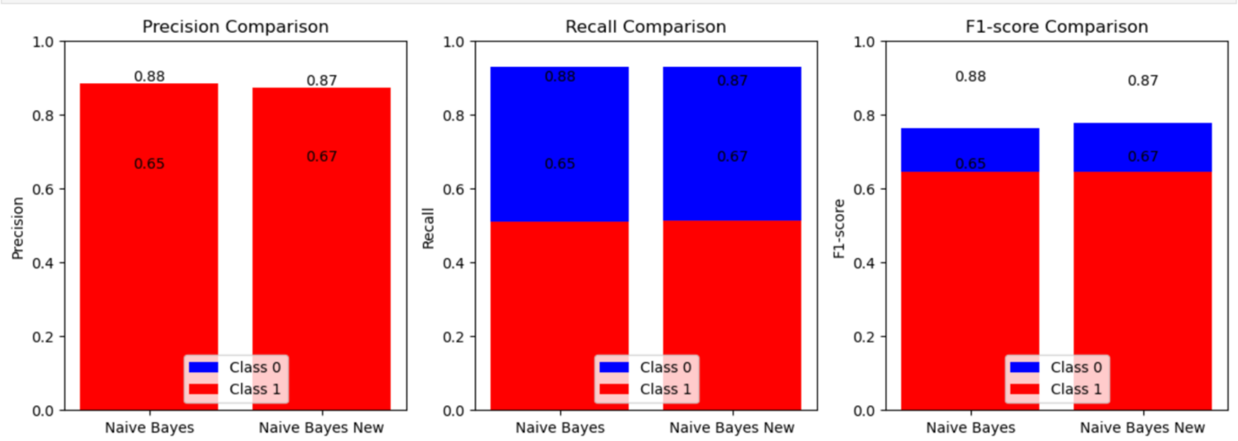
Best parameters - {'criterion': 'gini', 'max_depth': 19, 'n_estimators': 200}, Best score - 0.7421485570076316

Random forest Performance remains constant after hyper parameter tuning. Trying on huge number of combinations can increase the performance of random forest model.



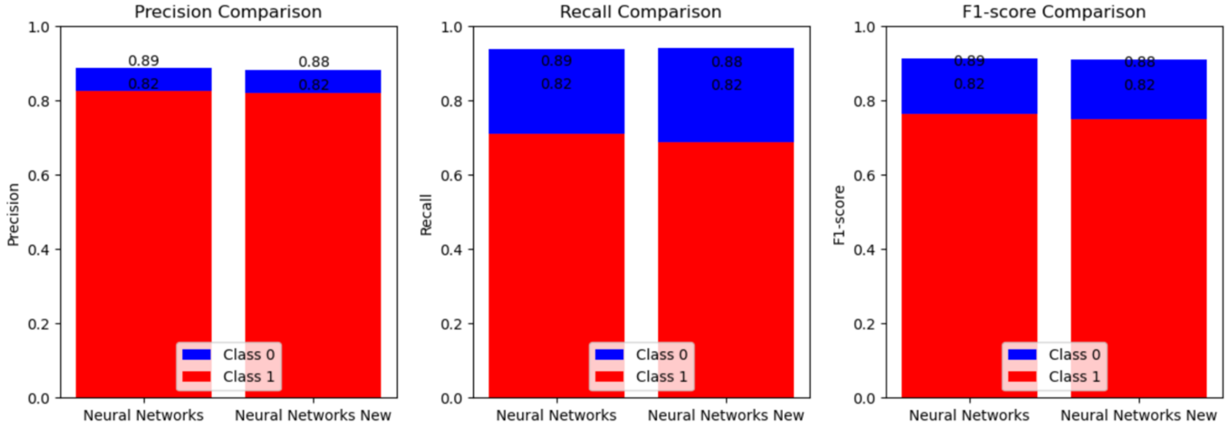
Naïve Bayes

It performed similar, even after outlier elimination.

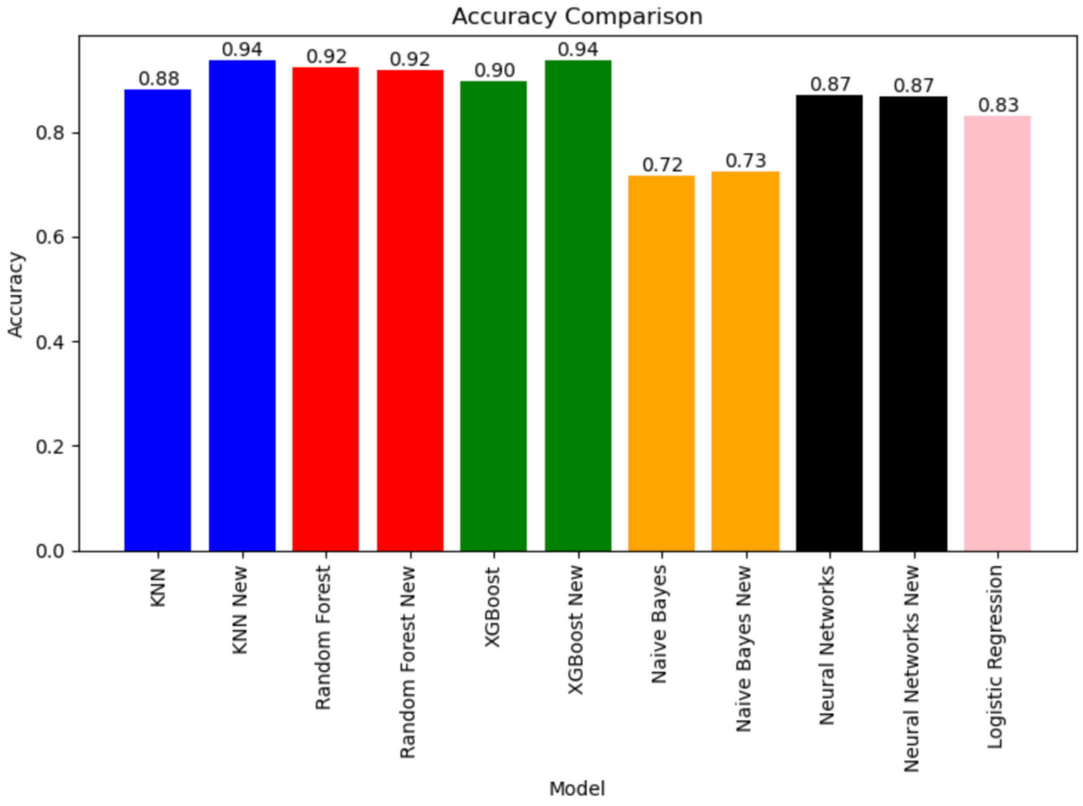


Neural Networks

Since, after manual hyper parameter tuning, neural networks performance was similar. Back propagation neural networks can be implemented to increase the model performance.

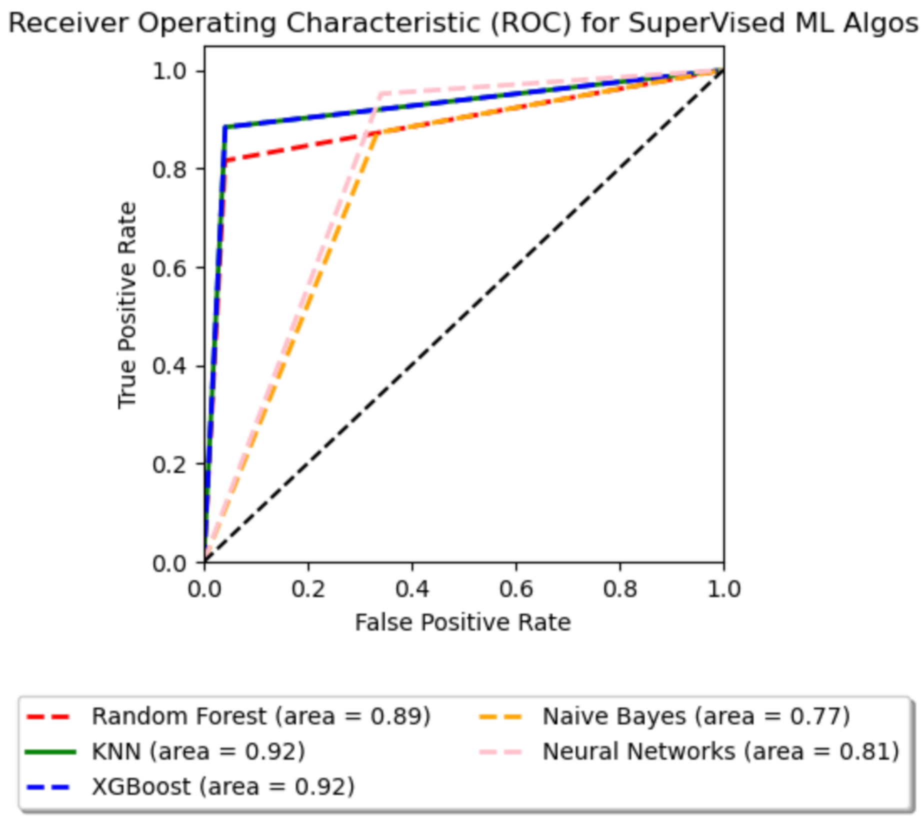


We can also compare the accuracy of various models.



ROC Curve

The Receiver Operating Characteristic (ROC) curve is a graphical representation used to assess the performance of a binary classification model. It plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings.
Best performance of the model lies in the top left corner of the plot.



Conclusion and Recommendations

Comparison of Model Performance:

Random Forest (RF): Achieved high accuracy, precision, recall, and F1-score for both old and new models. Known for its robust performance, particularly in handling noisy data and large feature sets. Requires minimal feature engineering and is less prone to overfitting compared to other complex models.

XGBoost (XGB): Demonstrated competitive performance, with slightly lower accuracy for the old model and improved accuracy for the new model. XGBoost is an ensemble learning method based on gradient boosting, which often yields state-of-the-art results in various machine learning competitions. Can handle missing data, multicollinearity, and non-linear relationships effectively.

Naive Bayes: Showed relatively showed not good performance. Naive Bayes assumes independence between features, which may not hold true in many real-world datasets. Despite its simplicity, Naive Bayes can perform well in text classification and other tasks with high-dimensional sparse data.

K-Nearest Neighbors (KNN): Achieved high performance with max accuracy compared to other models. It is a simple and intuitive algorithm but may suffer from computational inefficiency and the curse of dimensionality.

Neural Networks: Demonstrated competitive performance. Neural networks are highly flexible and capable of capturing complex patterns in data. They require extensive computational resources, hyperparameter tuning, and data preprocessing but can yield state-of-the-art results in various domains.

Computational Efficiency:

Random Forest (RF) and XGBoost (XGB): Both algorithms are known for their scalability and efficiency, especially for large datasets, due to their ability to compute in parallel. RF and XGB can handle large feature sets and datasets efficiently, making them right for production environments.

Naive Bayes: Naive Bayes is computationally efficient, as it only requires calculating simple probabilities based on the training data. It can be used to handle large datasets and is proper for real-time applications where computational efficiency is important.

K-Nearest Neighbors (KNN): KNN's computational complexity grows with the size of the training data, making it less efficient for large datasets. It requires storing the entire training dataset in memory, which can be memory-intensive for large datasets.

Neural Networks: Neural networks can be computationally intensive, especially during training, particularly for deep architectures and large datasets. Training neural networks often requires powerful hardware, such as GPUs, and may involve lengthy training times.

Applicability:

Random Forest (RF) and XGBoost (XGB): Suitable for a wide range of tasks, including classification and regression. RF and XGB often perform well with minimal feature engineering and can handle both numerical and categorical data effectively.

Naive Bayes: Suitable for text classification and other tasks with high-dimensional sparse data. Naive Bayes is commonly used in spam filtering, sentiment analysis, and document categorization tasks.

K-Nearest Neighbors (KNN): Suitable for small to medium-sized datasets and works well with feature scaling and low dimensionality. KNN is intuitive and easy to understand but may not scale well to large datasets or high-dimensional spaces.

Neural Networks: Versatile and applicable to various tasks, including image recognition, natural language processing, and time-series forecasting. Neural networks offer flexibility and can handle complex relationships in data but may require extensive computational resources and careful tuning.

References

Dataset: https://figshare.com/articles/dataset/S1_Data_-/24292936

Geeks for Geeks: https://www.geeksforgeeks.org/

Analytics Vidhya: https://www.analyticsvidhya.com/

Other: http://stackoverflow.com