# 574M Statistical Machine Learning

## Project Report

### Team: Kamaljeet Singh, Shrey Arora, Maheedhar Kolli

### Title: Huawei Digix - Advertisement CTR Prediction

## Introduction

Advertisement CTR prediction is the key problem in the area of computing advertising. Increasing the accuracy of Advertisement CTR prediction is critical to improve the effectiveness of precision marketing.

We analyze highly imbalanced dataset and deploy several classification models with different resampling techniques to minimize the effects of class imbalance on big advertising datasets that are anonymized.

## Data Overview

The datasets (after being masked) contain the advertising behavior data collected from seven consecutive days, including a training dataset and a testing dataset. This section describes the data format. Detailed data fields are illustrated in the following sections.

Below given table contains the column names of the training data along with description:

| Column name | Description |
|---|---|
| label | Label (0 = No Click, 1 = Click) |
| uid | Unique user ID after data anonymization |
| task_id | Unique ID of an ad task |
| adv_id | Unique ID of an ad material |
| creat_type_cd | Unique ID of an ad creative type |
| adv_prim_id | Advertiser ID of an ad task |
| dev_id | Developer ID of an ad task |
| inter_typ_cd | Display form of an ad material |
| slot_id | Ad slot ID |
| spread_app_id | App ID of an ad task |
| tags | App tag of an ad task |

| Column name | Description |
|---|---|
| app_first_class | App level-1 category of an ad task |
| app_second_class | App level-2 category of an ad task |
| age | User age |
| city | Resident city of a user |
| city_rank | Level of the resident city of a user |
| device_name | Phone model used by a user |
| device_size | Size of the phone used by a user |
| career | User occupation |
| gender | User gender |
| net_type | Network status when a behavior occurs |
| residence | Resident province of a user |
| his_app_size | App storage size |
| his_on_shelf_time | Release time |
| app_score | App rating score |
| emui_dev | EMUI version |
| list_time | Model release time |
| device_price | Device price |
| up_life_duration | HUAWEI ID lifecycle |
| up_membership_grade | Service membership level |
| membership_life_duration | Membership lifecycle |
| consume_purchase | Paid user tag |
| communication_onlinerate | Active time by mobile phone |
| communication_avgonline_30d | Daily active time by mobile phone |
| indu_name | Ad industry information |
| pt_d | Date when a behavior occurs |

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import imblearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
from xgboost.sklearn import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, precision_score, recall_score, roc_curve
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.feature_selection import f_classif,chi2
from sklearn import tree
import numpy as np
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import chi2_contingency
import random
import scipy.stats as stats

from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler

data = pd.read_csv("C:/Users/ual-laptop/Downloads/train_data.csv", sep = "|",nrows = 1
#data["label"].value_counts()
```

In [3]:

```
# Take random sample from thee data


columns = ['label', 'uid', 'task_id', 'adv_id', 'creat_type_cd', 'adv_prim_id',
       'dev_id', 'inter_type_cd', 'slot_id', 'spread_app_id', 'tags',
       'app_first_class', 'app_second_class', 'age', 'city', 'city_rank',
       'device_name', 'device_size', 'career', 'gender', 'net_type',
       'residence', 'his_app_size', 'his_on_shelf_time', 'app_score',
       'emui_dev', 'list_time', 'device_price', 'up_life_duration',
       'up_membership_grade', 'membership_life_duration', 'consume_purchase',
       'communication_onlinerate', 'communication_avgonline_30d', 'indu_name',
       'pt_d']



data = pd.DataFrame()

for i in range(1,51):
       #print(i)
       s = 10000*(i-1)
       df = pd.read_csv("C:/Users/ual-laptop/Downloads/train_data.csv", sep = "|",skip
       df.columns = columns
       skip = sorted(random.sample(range(10000),2400))
       df = df.iloc[skip]
       if i==1:
              data = df.copy()
       else:
              data = data.append(df,ignore_index=True)




data["label"].value_counts()
```

Out[3]:

```
0    115535
1      4465
Name: label, dtype: int64
```

# EDA

In the dataset we have a column label (Target variable) taking value 0 (add not clicked) and 1( add clicked) and 35 features including numerical features like app_score, membership_life_duration etc. and categorical features like gender, career etc. Data is being divided into continous and categorical variables.\

continous_cols = ["age","city_rank","device_size","his_app_size","his_on_shelf_time", "app_score","emui_dev","list_time","device_price","up_life_duration", "membership_life_duration","consume_purchase","communication_avgonline_30d","pt_d"]\
Categorical_cols = ['uid', 'task_id', 'adv_id', 'creat_type_cd', 'adv_prim_id', 'dev_id', 'inter_type_cd', 'slot_id', 'spread_app_id', 'tags', 'app_first_class', 'app_second_class', 'city', 'device_name', 'career', 'gender', 'net_type', 'residence', 'up_membership_grade', 'communication_onlinerate', 'indu_name'] \

From each of the categorical variables we have checked the total number of unique classes. If the number of classes are less than one percent of the dataset we have moved the categories to "others"

In [4]:
```
continous_cols = ["age","city_rank","device_size","his_app_size","his_on_shelf_time",
                  "app_score","emui_dev","list_time","device_price","up_life_duration"
                  "membership_life_duration","consume_purchase","communication_avgonli
categorical_cols = data.drop(columns = continous_cols).columns
categorical_cols = categorical_cols[1:len(categorical_cols)]
```

In [5]:
```
def give_count(colname):
    counts = pd.DataFrame()
    for i in set(data[colname]):
        counts= counts.append({"name":i,"count":len(data[data[colname==i]])},ignore_ir
    return counts
```

In [6]:
```
#ANOVA for continous variables

alpha = 0.01
f_values,p_values = f_classif(data[continous_cols], data["label"])
```

In [7]:
```
#Chisq test for categorical
Unique_values = []
for c in categorical_cols:
    Unique_values.append({c:len(set(data[c]))})
#Unique_values
```

## Chi-square test

We have performed chi-square test on the categorical columns to check the association between the categorical variables and output variable. The test result detects all variables have significant relationship between the variables.

In [8]:
```
p_values = pd.DataFrame()

for c in categorical_cols[1:len(categorical_cols)]:

    chisqt = pd.crosstab(data["label"], data[c], margins=True)

    value = np.array([chisqt.iloc[0][0:len(set(data[c]))+2].values,
                      chisqt.iloc[1][0:len(set(data[c]))+2].values])
    c = chi2_contingency(value)[0:3]
    d = {"statistic":c[0],"p-value":c[1],"dof":c[2]}
```

```
    p_values = p_values.append(d,ignore_index=True)

p_values["column"] = categorical_cols[1:len(categorical_cols)]
```

In this section, we introduced dummy variables for all the categorical variables retained from EDA.

In [9]:
```
encoded_data = pd.get_dummies(data, columns = ['creat_type_cd', 'adv_prim_id',
       'dev_id', 'inter_type_cd', 'slot_id', 'spread_app_id', 'tags',
       'app_first_class', 'app_second_class', 'city', 'device_name', 'career',
       'gender', 'net_type', 'residence', 'up_membership_grade',
        'indu_name'])
```

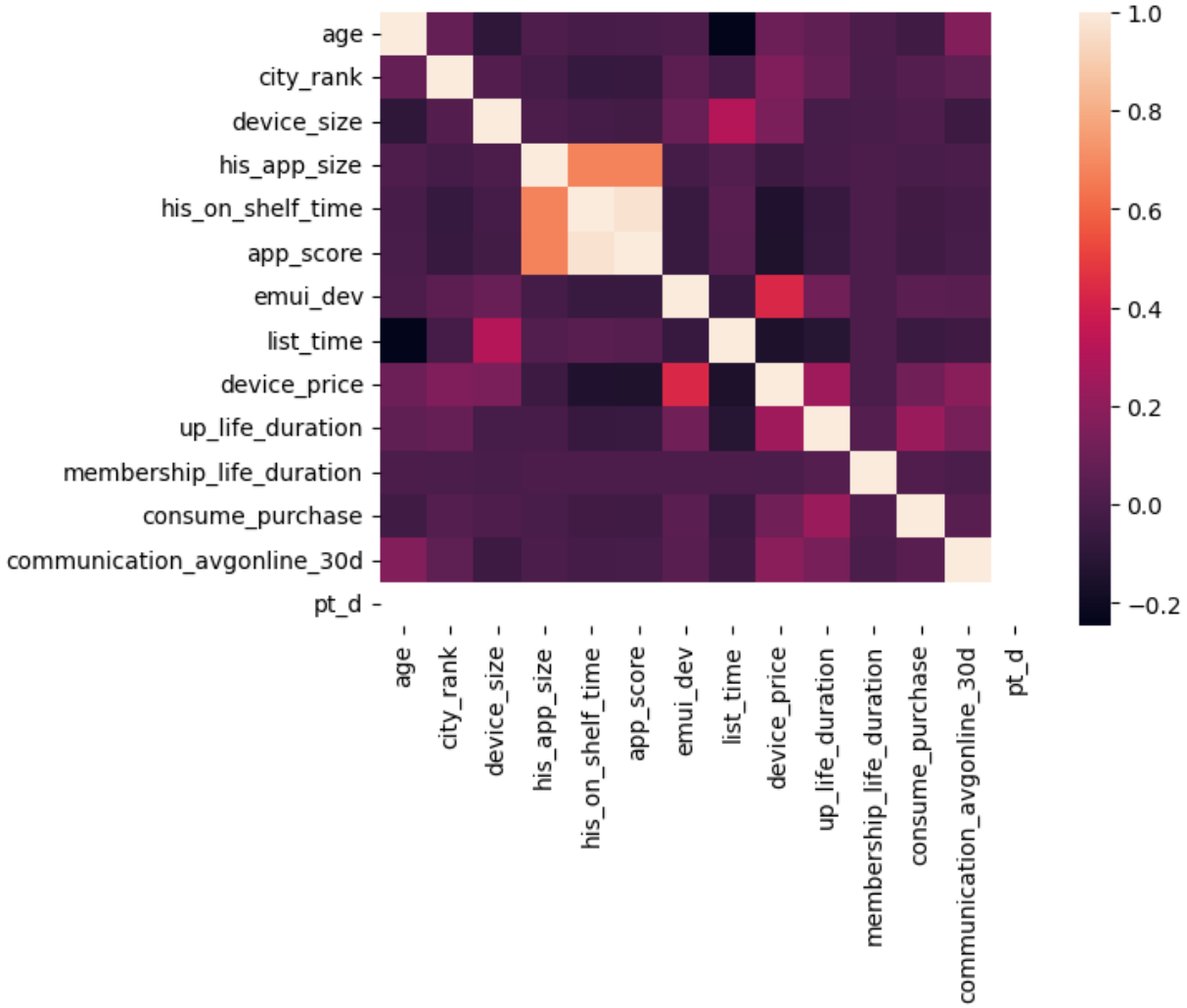
In [10]:
```
X = encoded_data.drop(columns = ["label"])
y = encoded_data["label"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

## Heat map

The heat map on the dataset is visualized below

In [11]:
```
sns.heatmap(X_train[continous_cols].corr())
```

Out[11]:
```
<AxesSubplot:>
```




From above heat map we can observe that the columns “his_app_size”, “his_on_shelf_time”, “app_score” are correlated to each other.

# Feature engineering

For the variable Communication_onlinerate which takes value of the form 3^4^5^6^7^8^9^10^11^12^13^14^15^16^17^18^19^20 and shows the active time of a user on his/her device, we extracted the starting and ending time and constructed two separate variables for starting and ending time.

In [12]:
```
def convert_communication_onlinerate1(row):
    return int(row["communication_onlinerate"].split("^")[0])

def convert_communication_onlinerate2(row):
    return int(row["communication_onlinerate"].split("^")[-1])
```

In [13]:
```
X_train["communication_onlinerate_start"] = X_train.apply( convert_communication_onlir
X_train["communication_onlinerate_end"] = X_train.apply( convert_communication_onliner
X_test["communication_onlinerate_start"] = X_test.apply( convert_communication_onliner
X_test["communication_onlinerate_end"] = X_test.apply( convert_communication_onlinerat
```

```
X_train1 = X_train.drop(columns = ["communication_onlinerate","pt_d","uid","task_id","
X_test1 = X_test.drop(columns = ["communication_onlinerate","pt_d","uid","task_id","ad
```

But after this operation, we were getting 4000+ variables and it was very difficult for us to deal with large number of variables in terms of computation cost. To deal with this, we decided to create a separate category (say other) for each of the categorical variables and all the categories occurring less than 1% of the time are grouped in the other category. In this manner, the total number of variables reduced to 462.

# Baseline Models

# Decision Tree

Since this is a classification problem with a large number of categorical variables, our first choice was to use Decision Tree. Since the dataset is highly imbalanced, the initial accuracy (precision and recall) is not very good. So, we decided to apply Random Forest algorithm.

In [14]:
```
from sklearn.metrics import classification_report
model = DecisionTreeClassifier()
model.fit(X_train1,y_train)
print(classification_report(y_test,model.predict(X_test1)))
```

```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96     23084
           1       0.07      0.06      0.07       916

    accuracy                           0.93     24000
   macro avg       0.51      0.51      0.51     24000
weighted avg       0.93      0.93      0.93     24000
```

# Random Forest

Although it is expected that Random Forest will provide better results than Decision Tree but, in our case, there isn't any significant improvement over Decision Tree.

In [36]:
```
m2 = RandomForestClassifier()
m2.fit(X_train1,y_train)
```

Out[36]:
```
RandomForestClassifier()
```

In [37]:
```
print(classification_report(m2.predict(X_test1),y_test))
```

```
              precision    recall  f1-score   support

           0       1.00      0.96      0.98     23966
           1       0.01      0.32      0.02        34

    accuracy                           0.96     24000
   macro avg       0.51      0.64      0.50     24000
weighted avg       1.00      0.96      0.98     24000
```

# XGBoost

After Decision Tree and Random Forest, we tried XGBoost, but due to highly imbalanced data, even XGBoost could not provide any good result.

In [17]:
```
m3 = xgb.XGBClassifier(use_label_encoder=False)
m3.fit(X_train1,y_train)
print(classification_report(y_test,m3.predict(X_test1)))
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98     23084
           1       0.40      0.01      0.01       916

    accuracy                           0.96     24000
   macro avg       0.68      0.50      0.50     24000
weighted avg       0.94      0.96      0.94     24000
```

# Logisitc

In [18]:
```
log = LogisticRegression(penalty='l1',C=0.01, solver = "saga")
#scores = cross_val_score(log, X_train1, y_train, cv=5, scoring='precision')
log.fit(X_train1,y_train)
print(classification_report(y_test,log.predict(X_test1)))
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98     23084
           1       0.00      0.00      0.00       916

    accuracy                           0.96     24000
   macro avg       0.48      0.50      0.49     24000
weighted avg       0.93      0.96      0.94     24000
```

# SMOTE

## Undersampling

In this technique, we tried to under sample the 0 class labels to balance the data. We resampled the data for different value of r between 0 and 0.5, where r = no of label 1/no of label 0. We

observed that precision and recall vary significantly with the value of r as shown in the plot.

In [28]:
```
#Under sampling
rus = RandomUnderSampler(random_state=42, replacement=True)# fit predictor and target
x_rus, y_rus = rus.fit_resample(X_train1, y_train)
```

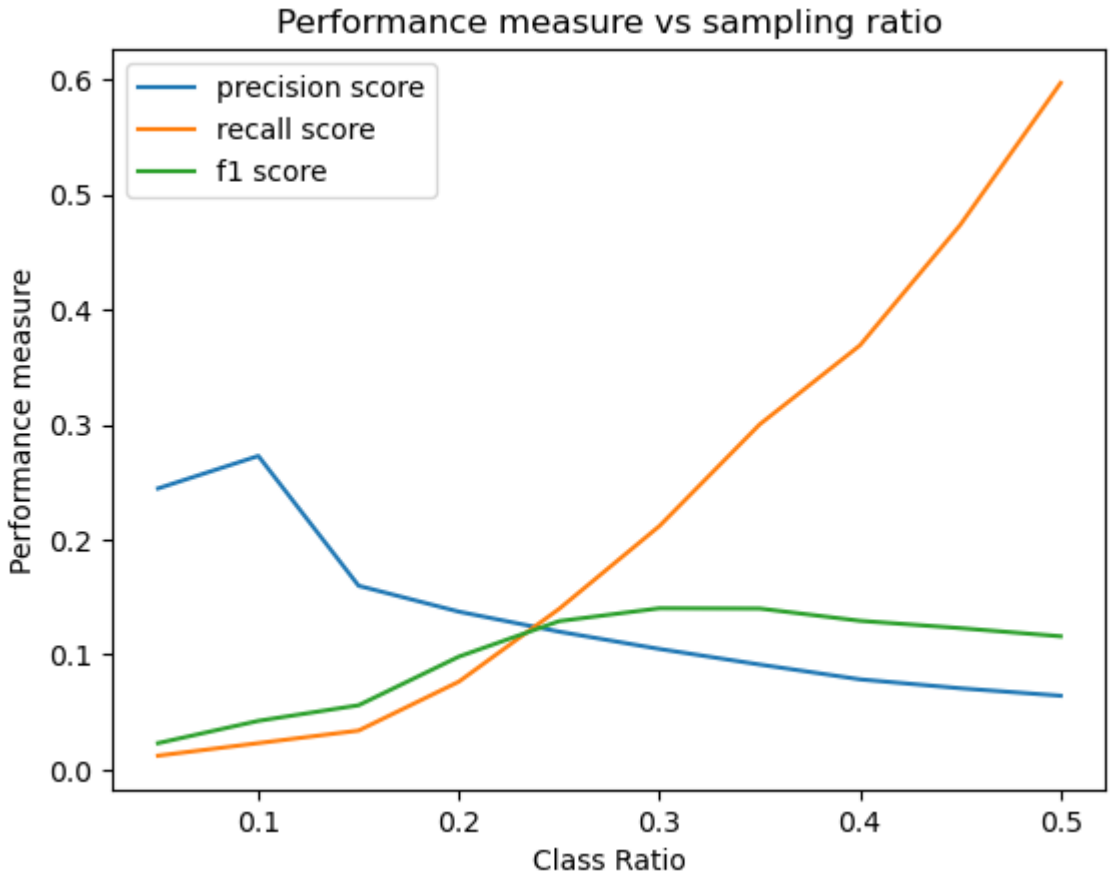
In [29]:
```
ratio = [0.05,0.1,0.15,0.20,0.25,0.30,0.35,0.4,0.45,0.5]
n1 = 92521
n2 = 3479
precision_scores = []
recall_scores = []

x1 = X_train1[y_train==1]
x0 = X_train1[y_train==0]
y1 = y_train[y_train==1]
```

In [30]:
```
for r in ratio:


    sample_size =  int(float(n2)*(1/r-1))
    subset_index= sorted(random.sample(range(0,len(x0)),sample_size))

    x0 = x0.iloc[subset_index]
    x_rus = x1.append(x0)
    y0 = y_train[y_train==0].iloc[subset_index]
    y_rus = y1.append(y0)


    #len(x_rus.columns)
    #models = [ DecisionTreeClassifier(),RandomForestClassifier(),xgb.XGBClassifier(us
    m = RandomForestClassifier()
    m.fit(x_rus,y_rus)
    precision_scores.append(precision_score(y_test,m.predict(X_test1)))
    recall_scores.append(recall_score(y_test,m.predict(X_test1)))
```

In [22]:
```
f1_scores = []
for i in range(0,10):
    f1_scores.append(2*precision_scores[i]*recall_scores[i]/(precision_scores[i]+recal
```

In [23]:
```
plt.plot(ratio,precision_scores,label="precision score")
plt.plot(ratio,recall_scores, label="recall score")
plt.plot(ratio,f1_scores, label="f1 score")
plt.title("Performance measure vs sampling ratio")
plt.xlabel("Class Ratio")
plt.ylabel("Performance measure")
plt.legend(loc="upper left")
```

Out[23]:
```
<matplotlib.legend.Legend at 0x262bde20fa0>
```




# Decision Tree

In [24]:
```
#Tree
model = DecisionTreeClassifier()
model.fit(x_rus,y_rus)
print(classification_report(y_test,model.predict(X_test1)))
```

```
              precision    recall  f1-score   support

           0       0.97      0.56      0.71     23084
           1       0.05      0.58      0.09       916

    accuracy                           0.56     24000
   macro avg       0.51      0.57      0.40     24000
weighted avg       0.94      0.56      0.68     24000
```

# Random Forest

In [25]:
```
m2 = RandomForestClassifier()
m2.fit(x_rus,y_rus)
print(classification_report(y_test,m2.predict(X_test1)))
```

```
              precision    recall  f1-score   support

           0       0.98      0.64      0.78     23084
           1       0.06      0.59      0.11       916

    accuracy                           0.64     24000
   macro avg       0.52      0.62      0.44     24000
weighted avg       0.94      0.64      0.75     24000
```

# XGBoost

In [26]:
```
#XGBoost
x1_rus, y_rus = rus.fit_resample(X_train1, y_train)
m3 = xgb.XGBClassifier(use_label_encoder=False)
m3.fit(x1_rus,y_rus)
print(classification_report(y_test,m3.predict(X_test1)))
```

```
              precision    recall  f1-score   support

           0       0.98      0.63      0.77     23084
           1       0.06      0.60      0.11       916

    accuracy                           0.63     24000
   macro avg       0.52      0.62      0.44     24000
weighted avg       0.94      0.63      0.74     24000
```

# Oversampling using SMOTE

SMOTE (Synthetic Minority Oversampling Technique) is a technique used to address class imbalance in a dataset. Class imbalance refers to a situation in which one class in a classification dataset has significantly more examples than another class. This can be a problem because many machine learning algorithms are designed to maximize accuracy, and they can sometimes have difficulty learning from imbalanced datasets. SMOTE works by creating new synthetic examples of the minority class in order to balance the dataset. It does this by selecting a minority class example and finding its nearest neighbors. It then creates new examples that are some combination of the selected example and its neighbors. This can help to improve the performance of machine learning algorithms on imbalanced datasets. In the data set after applying this technique we can observe that the classes have been almost balanced. Then we applied the model, here are the following results

In [31]:
```
# oversampling
from imblearn.over_sampling import SMOTE, ADASYN
# fit predictor and target variable
x_ros, y_ros = ADASYN().fit_resample(X_train1, y_train)
```

# Decision Tree

In [32]:
```
model = DecisionTreeClassifier()
model.fit(x_ros,y_ros)
```

```
print(classification_report(y_test,model.predict(X_test1)))
```

```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96     23084
           1       0.06      0.06      0.06       916

    accuracy                           0.93     24000
   macro avg       0.51      0.51      0.51     24000
weighted avg       0.93      0.93      0.93     24000
```

# Random Forest

In [33]:
```
m2 = RandomForestClassifier()
m2.fit(x_ros,y_ros)
print(classification_report(y_test,m2.predict(X_test1)))
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98     23084
           1       0.25      0.02      0.04       916

    accuracy                           0.96     24000
   macro avg       0.61      0.51      0.51     24000
weighted avg       0.94      0.96      0.94     24000
```

# XGBoost

In [34]:
```
x1_ros, y_ros = ADASYN().fit_resample(X_train1, y_train)
m3 = xgb.XGBClassifier(use_label_encoder=False)
m3.fit(x1_ros,y_ros)
print(classification_report(y_test,m3.predict(X_test1)))
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98     23084
           1       0.23      0.02      0.04       916

    accuracy                           0.96     24000
   macro avg       0.60      0.51      0.51     24000
weighted avg       0.93      0.96      0.94     24000
```

# Logistic Regression

In [35]:
```
log = LogisticRegression(penalty='l1',C=0.01, solver = "saga")
log.fit(x_ros,y_ros)
print(classification_report(y_test,log.predict(X_test1)))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97     23084
           1       0.12      0.06      0.08       916

    accuracy                           0.95     24000
   macro avg       0.54      0.52      0.53     24000
weighted avg       0.93      0.95      0.94     24000
```

# Results

From the above models we can observe that after resampling better results can be seen when the data is undersampled. As the data set is highly imbalanced high precision and recall cannot be attained simultaneously hence it depends on the business needs which is a tradeoff. The best results are summarized in the below table.

| Model | precision | Recall | F1-Score | Technique |
|---|---|---|---|---|
| Decision Tree | 0.05 | 0.58 | 0.09 | Undersampling |
| Random Forest | 0.06 | 0.59 | 0.11 | Undersampling |
| XG Boost | 0.06 | 0.60 | 0.11 | Undersampling |
| Logistic Regression | 0.12 | 0.06 | 0.08 | Oversampling |