**Task 3: Model Training and Evaluation**

**1.Logistic Regression (Before Resampling)**

1. import pandas as pd
2. from sklearn.model\_selection import train\_test\_split as tts
3. from sklearn.linear\_model import LogisticRegression
4. from sklearn.impute import SimpleImputer
5. # Assuming df1 is your dataframe
6. # 1. Imputation: Replace NaN with a specific value (e.g., mean, median, or a constant).
7. imputer = SimpleImputer(strategy='mean')  # You can change the strategy
8. x\_ = df1.drop(['target', 'id'], axis=1)
9. x\_ = pd.DataFrame(imputer.fit\_transform(x\_), columns=x\_.columns) # Fit and transform on x\_
10. y\_ = df1['target']
11. # 2. Split data after handling missing values
12. X\_Train, X\_Test, y\_Train, y\_Test = tts(x\_, y\_, test\_size=0.3, random\_state=1)

**Logistic Regression (After Resampling)**

1. Fitting a Logistic Regression

1. # Proceed with model training
2. log\_reg = LogisticRegression()
3. log\_reg.fit(X\_Train, y\_Train)
4. y\_pred\_logreg1 = log\_reg.predict(X\_Test)
5. # Assuming accuracy\_result is your function to calculate accuracy
6. # accuracy\_result(y\_Test, y\_pred\_logreg1)

**2. Logistic Regression (After Resampling)**

1. Create an imputer to replace NaN values with the mean of the column

imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median' or 'most\_frequent'

# 2. Fit the imputer on your training data and transform it

X\_train\_imputed = imputer.fit\_transform(X\_train)

# 3. Now, you can use the imputed data for training your model

log\_reg = LogisticRegression()

log\_reg.fit(X\_train\_imputed, y\_train)

# Remember to apply the same imputation to your test data before prediction:

X\_test\_imputed = imputer.transform(X\_test)

y\_pred\_logreg2 = log\_reg.predict(X\_test\_imputed)

**Q. What is the accuracy score and f1-score for the improved Logistic Regression model?**

**Q. Why do you think f1-score has improved?**

**Ans.:**

**Logistic regression model (Before Resampling)**

* Accurecy = 0.964
* F1 score = 0.0

**Logistic regression model (After Resampling)**

* Accurecy = 0.589
* F1 score = 0.573

In above given data after resampling the F1 score is 0.573 which is improved F1 score in compare to previous score and it shows improved logistic regression model.

**3. Support Vector Classifier**

x\_tr=X\_train[:10000,:]  
y\_tr=y\_train[:10000]  
  
## 1. svm Classifier with linear kernel  
  
#Create a svm Classifier  
clf = svm.SVC(kernel='linear')  
# FIT SVC ON TRAINING DATA  
clf.fit(x\_tr, y\_tr)  
  
## 2. Predicting on Train and test data  
  
# y\_pred\_train = clf.predict(X\_train)  
y\_pred\_svc = clf.predict(X\_test)  
  
#  model output  
accuracy\_result(y\_test, y\_pred\_svc)

CONFUSION MATRIX:

[[17223 9653]

[12480 14149]]

FPR: 0.359

TPR/ RECALL/ SENSTIVITY: 0.531

PRECISION: 0.594

SPECIFICITY: 0.641

ACCURACY: 0.586

ROC AUC: 0.586

Cohens kappa: 0.172

F1 score: 0.561

CLASSIFICATION REPORT:

precision recall f1-score support

0 0.58 0.64 0.61 26876

1 0.59 0.53 0.56 26629

accuracy 0.59 53505

macro avg 0.59 0.59 0.58 53505

weighted avg 0.59 0.59 0.59 53505

**Q.: For model LinearSVC play with parameters – dual, max\_iter and see if there is any improvement.**

Ans.: Not any Significance Improvement

**Q.: SVC with Imbalance Check & Feature Optimization & only 100K Records → is there improvement in scores?**

Ans.: I fitted the SVC model on 10K dataset because on 100K dataset not able to run the model.

**4. XGBoost Classifier**

## 1. Initializing XGboost Classifier  
  
xg\_cls = XGBClassifier(objective='binary:logistic', colsample\_bytree = 0.3, learning\_rate = 0.1,  
                max\_depth = 50, alpha = 10, n\_estimators = 100)  
xg\_cls.fit(X\_train, y\_train)  
  
## 2. Predicting on Train and test data  
  
# y\_pred\_train = xg\_cls.predict(X\_train)  
y\_pred\_xgb = xg\_cls.predict(X\_test)  
  
# function for evaluating the model output  
accuracy\_result(y\_test, y\_pred\_xgb)

CONFUSION MATRIX:

[[26795 81]

[ 14 26615]]

FPR: 0.003

TPR/ RECALL/ SENSTIVITY: 0.999

PRECISION: 0.997

SPECIFICITY: 0.997

ACCURACY: 0.998

ROC AUC: 0.998

Cohens kappa: 0.996

F1 score: 0.998

CLASSIFICATION REPORT:

precision recall f1-score support

0 1.00 1.00 1.00 26876

1 1.00 1.00 1.00 26629

accuracy 1.00 53505

macro avg 1.00 1.00 1.00 53505

weighted avg 1.00 1.00 1.00 53505

**Q. XGBoost is one the better classifiers -- but still f1-score is very low. What could be the reason?**

In my model ,I'm getting f1 score near to 1, that indicate a best fitted model.

**Q. What is the increase in number of features after one-hot encoding of the data?**

Ans.: There is no need of One-hot encoding because it has been previously done.

**Q. Is there any improvement in scores after encoding?**

Ans.: Not happened (because There is no need of One-hot encoding so I have not done it)

**Q. If not missing a positive sample is the priority which model is best so far?**

Ans.: XGBoost model is performing best. Count of missing positive sample is 6 only.

\*\*XGBoost CONFUSION MATRIX:  
  
 [[172287      6]  
  
 [     0 171818]]

**Q. If not marking negative sample as positive is top priority, which model is best so far?**

Ans.: XGBoost model is performing best. No misclassification for negative sample as positive.

 \*\*XGBoost CONFUSION MATRIX:  
  
 [[172287      6]  
  
 [     0 171818]]

**5. Adaboost classifier**

# Create adaboost classifer object  
adaBoost = AdaBoostClassifier(n\_estimators=50, learning\_rate=1, random\_state=0)  
  
# Create an imputer to replace NaN values with the mean of each column  
imputer = SimpleImputer(strategy='mean')    
  
# Fit the imputer on your training data and transform both train and test data  
X\_train\_imputed = imputer.fit\_transform(X\_train)  
X\_test\_imputed = imputer.transform(X\_test)  
  
# Train Adaboost Classifer using the imputed data  
model1 = adaBoost.fit(X\_train\_imputed, y\_train)  
  
#Predict the response for test dataset using the imputed data  
y\_pred\_ada = model1.predict(X\_test\_imputed)  
  
accuracy\_result(y\_test, y\_pred\_ada)

CONFUSION MATRIX:

[[16866 10010]

[11052 15577]]

FPR: 0.372

TPR/ RECALL/ SENSTIVITY: 0.585

PRECISION: 0.609

SPECIFICITY: 0.628

ACCURACY: 0.606

ROC AUC: 0.606

Cohens kappa: 0.213

F1 score: 0.597

CLASSIFICATION REPORT:

precision recall f1-score support

0 0.60 0.63 0.62 26876

1 0.61 0.58 0.60 26629

accuracy 0.61 53505

macro avg 0.61 0.61 0.61 53505

weighted avg 0.61 0.61 0.61 53505

**Q.: Do you think using AdaBoost can give any significant improvement over XGBoost?**

**Ans.:** No improvement in AdaBoost model result because In my case XGBoost model giving much more better result (Accuracy and F1 score) in compare to AdaBoost model.

**6. MLP Classifier**

import pandas as pd  
from sklearn.neural\_network import MLPClassifier  
from sklearn.impute import SimpleImputer  
  
# Create an imputer to replace NaN values with the mean of each column  
imputer = SimpleImputer(strategy='mean')    
  
# Fit the imputer on your training data and transform both train and test data  
X\_train\_imputed = imputer.fit\_transform(X\_train)  
X\_test\_imputed = imputer.transform(X\_test)  
  
# Training the model using the imputed data  
mlp = MLPClassifier(hidden\_layer\_sizes=(10, 10, 10), activation='logistic', max\_iter=200)  
mlp.fit(X\_train\_imputed, y\_train.values.ravel()) # Use imputed data here  
  
# Prediction on x\_test using the imputed data  
y\_predic = mlp.predict(X\_test\_imputed) # Use imputed data here  
accuracy\_result(y\_test, y\_predic)

CONFUSION MATRIX:

[[12754 14122]

[ 3270 23359]]

FPR: 0.525

TPR/ RECALL/ SENSTIVITY: 0.877

PRECISION: 0.623

SPECIFICITY: 0.475

ACCURACY: 0.675

ROC AUC: 0.676

Cohens kappa: 0.351

F1 score: 0.729

CLASSIFICATION REPORT:

precision recall f1-score support

0 0.80 0.47 0.59 26876

1 0.62 0.88 0.73 26629

accuracy 0.67 53505

macro avg 0.71 0.68 0.66 53505

weighted avg 0.71 0.67 0.66 53505

**Q.: MLPClassifier is the neural network we are trying. But how to choose the right no. of layers and size?**

**Q.: At what layer size we get the best f1-score?**

**Ans.:**

In Generally, you can't analytically calculate the number of layers or the number of nodes to use per layer in an ANN to address a specific real-world predictive modeling problem, But there are many methods for determining the correct number of neurons to use in the hidden layers, such as the following:

* The no. of hidden neurons should be between the size of the input layer and the size of the output layer.
* The no. of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
* The no. of hidden neurons should be less than twice the size of the input layer.

**Task 4: Model Optimization**

**Final Result Comparison**

data = {'Log\_Reg(Before Resamp)':[np.round(f1\_score(y\_Test, y\_pred\_logreg1),3), np.round(metrics.accuracy\_score(y\_Test, y\_pred\_logreg1),3)],  
        'Log\_Reg(After Resamp)':[np.round(f1\_score(y\_test, y\_pred\_logreg2),3), np.round(metrics.accuracy\_score(y\_test, y\_pred\_logreg2),3)],  
        'SVC':[np.round(f1\_score(y\_test, y\_pred\_svc),3), np.round(metrics.accuracy\_score(y\_test, y\_pred\_svc),3)],  
        'XGBoost':[np.round(f1\_score(y\_test, y\_pred\_xgb),3), np.round(metrics.accuracy\_score(y\_test, y\_pred\_xgb),3)],  
        'AdaBoost':[np.round(f1\_score(y\_test, y\_pred\_ada),3), np.round(metrics.accuracy\_score(y\_test, y\_pred\_ada),3)],  
        'MLP Classifier':[np.round(f1\_score(y\_test, y\_predic),3), np.round(metrics.accuracy\_score(y\_test, y\_predic),3)]}  
  
df\_result=pd.DataFrame(data, index = ['F1\_score', 'Accuracy']).T  
df\_result

Next steps:Generate code withdf\_resulttoggle\_offView recommended plotsNew interactive sheet

**After comparing the F1- score and Accuracy of all used Machine Learning models, we found that XGBoost model is best performing and giving best result with respect to all other model.**