OPTIMIZING DETECTION OF CREDIT CARD FRAUD USING MACHINE LEARNING TECHNIQUES

Report submitted to SASTRA Deemed to be University As per the requirement for the course

CSE300: MINI PROJECT

Submitted by

SUDHARSHANAN S
(125003354, B. Tech Computer Science and Engineering)

KRISHAANT S H
(125003153, B. Tech Computer Science and Engineering)

KARTHIKEYAN S
(125003137, B. Tech Computer Science and Engineering)

May 2023

SCHOOL OF COMPUTING

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act,1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

THANJAVUR, TAMIL NADU, INDIA – 613 401

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the report titled **"Optimizing Detection Of Credit Card Fraud Using Machine Learning Techniques"** submitted as a requirement for the course, **CSE300 : MINI PROJECT** for B.Tech. is a bonafide record of the work done by **Mr. SUDHARSHANAN S (125003354, B. Tech Computer Science and Engineering**), **Mr. KRISHAANT S H (125003153, B. Tech Computer Science and Engineering**) and **Mr. KARTHIKEYAN S(125003137, B. Tech Computer Science and Engineering)** during the academic year 2022-23, in the School of Computing, under my supervision.

Signature of Project Supervisor :

Name with Affiliation :

Date :

Mini Project *Viva voice* held on _____________

Examiner 1 **Examiner 2**

ACKNOWLEDGEMENTS

We would like to thank our Honorable Chancellor **Prof. R SETHURAMAN** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honorable Vice-Chancellor **Dr. S VAIDHYASUBRAMANIAM** and **Dr. S SWAMINATHAN**, Dean, Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R CHANDRAMOULI**, Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr. SHANKAR SRIRAM V S**, Dean School of Computing, **Dr. MUTHAIAH R**, Associate Dean - Research, **Dr. RAMUMAR K**, Associate Dean – Academics, **Dr. MANIVANNAN D**, Associate Dean - Infrastructure, **Dr. ALAGESWARAN R**, Associate Dean – Student Welfare.

Our guide **Ms. HELEN W R**, Asst. Professor - III, School of Computing was the driving force behind this whole idea from the start. Her deep insight in the field and invaluable suggestions helped us in making progress throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. We thank you all for providing me an opportunity to showcase my skills through this project.

List of Figures

List of Tables

Abbreviations

DT	Decision Tree
ET	Extra Trees
GBC	Gradient Boosting Classifier
KNN	K-Nearest Neighbor
LDA	Linear Discriminant Analysis
LR	Linear Regression
NB	Naive Bayes
QDA	Quadratic Discriminant Analysis
RF	Random Forest

ABSTRACT

Fraud involves criminal deception and false representations to gain an unfair advantage, particularly amplified by the growth in online transactions and technologies. The widespread use of online transaction systems and IoT devices has increased transaction volumes, heightening the risk of fraudulent activities. Given the prevalence of fraud, there is an urgent call for effective fraud detection systems.

In general, fraud detection can be categorized into two types: misuse detection and anomaly detection. Misuse detection involves the use of machine learning-based classification models to distinguish between fraudulent and legitimate transactions. On the other hand, anomaly detection establishes a baseline from sequential records to define the characteristics of a typical transaction and creates a distinctive profile for it.

We proposed a strategy for misuse detection that utilizes a combination of K-nearest neighbor (KNN), linear discriminant analysis (LDA), and linear regression (LR) models. Then we enhance the results with few modifications. The features extracted using this strategy demonstrated recall scores higher values across four tested fraud datasets. As a result, this approach surpasses other methods that rely on single machine learning models, particularly in terms of recall.

KEY WORDS: Recall scores, Fraud detection systems, K-nearest neighbor (KNN), Linear discriminant analysis (LDA), Linear regression (LR).

Table of Contents

CHAPTER 1
SUMMARY OF THE BASE PAPER

Title	**:**	Credit Card Fraud Detection: An Improved Strategy for High Recall Using KNN, LDA, and Linear Regression
Publisher	**:**	MDPI
Year	**:**	2023
Journal name	**:**	Sensors
DOI	**:**	https://doi.org/10.3390/s23187788
Base paper URL	**:**	https://www.mdpi.com/1424-8220/23/18/7788

The main contributions of the base paper are:

- Detecting Fraudulent Credit Card Transactions
- Combining multiple models to get a refined and exact results
- Improving the results with a proposed algorithms which combines 3 models result.

Our method consists of 3 major steps:

1.1. Data Cleaning :

The number of missing values per column and the median of missing values across columns are calculated. Then columns with missing values exceeding the median are removed. Missing values in the remaining columns are imputed with the median of each column. Encoding was performed for categorical attributes.

1.2 Balancing the Dataset Using SMOTE method:

SMOTE (Synthetic Minority Oversampling Technique) is a technique that resolves irregular classes in data by adding minority classes. This technique helps to balance the dataset by generating synthetic values for minority class .

Algorithm :

1. Identify the Minority Class: Determine the minority class, which has fewer instances than the majority class.
1. Calculate k-Nearest Neighbours : For each instance in the minority class, find its k-nearest neighbors based on a distance metric, typically Euclidean distance.
2. Create Synthetic Instances: For each minority class instance, randomly select one of its k-nearest neighbours. Then, create a synthetic instance by interpolating along the line segment between the original instance and the selected neighbour.

3. Add Synthetic Instances: Add the generated synthetic instances to the dataset, expanding the size of the minority class.
4. Repeat Process: Continue creating synthetic instances until the equal number of values are obtained .

1.3. Model training and algorithm testing

The models that have been used in this project are **K-nearest neighbour (KNN) , Linear Discriminant Analysis (LDA)** and **Linear Regression (LR) .** The hyperparameter set for each models are :

KNN :
algorithm='auto'
leaf_size=30
metric='minkowski'
metric_params=None
n_jobs=-1
n_neighbors=5
p=2
weights='uniform'

LDA :
covariance_estimator=None
n_components=None
priors=None
shrinkage=None
solver='svd'
store_covariance=False
tolerance=0.0001

LR:
Default settings.

All the categorical attributes were encoded using Label-Encoder and One hot encoding. The attributes are
Dataset 1 : Label Encoder was used
- 'type', 'nameOrig', and ‘nameDest'

Dataset 2: Label Encoder was used
- "merchant", "category", "first", "last", "gender", "street", "job", "trans_num", "city", "state", and “dob"

Dataset 3: Label Encoder was used
- gender”, “car”, “reality”, “income_type”, “education_type”, “house_type”, and “family_type

Dataset 4 :
ProductCD, card1 - card6, addr1, addr2, P_emaildomain, R_emaildomain , M1 - M9 .
In data preprocessing the missing values were filled using the median of that attributes and if the number of missing values exceeded a threshold value, then that attribute is removed.
To balance the dataset **SMOTE** has been used. So, the minority class was oversampled and the models were fitted using these values.

Splitting the Datasets:

All the datasets were split into train and test data using **Stratified K-Fold cross-validation** and the fold value was set to 5 .

Model Training:

The 3 models were fitted for the 4 datasets and the output result was given to the algorithm.

Algorithm

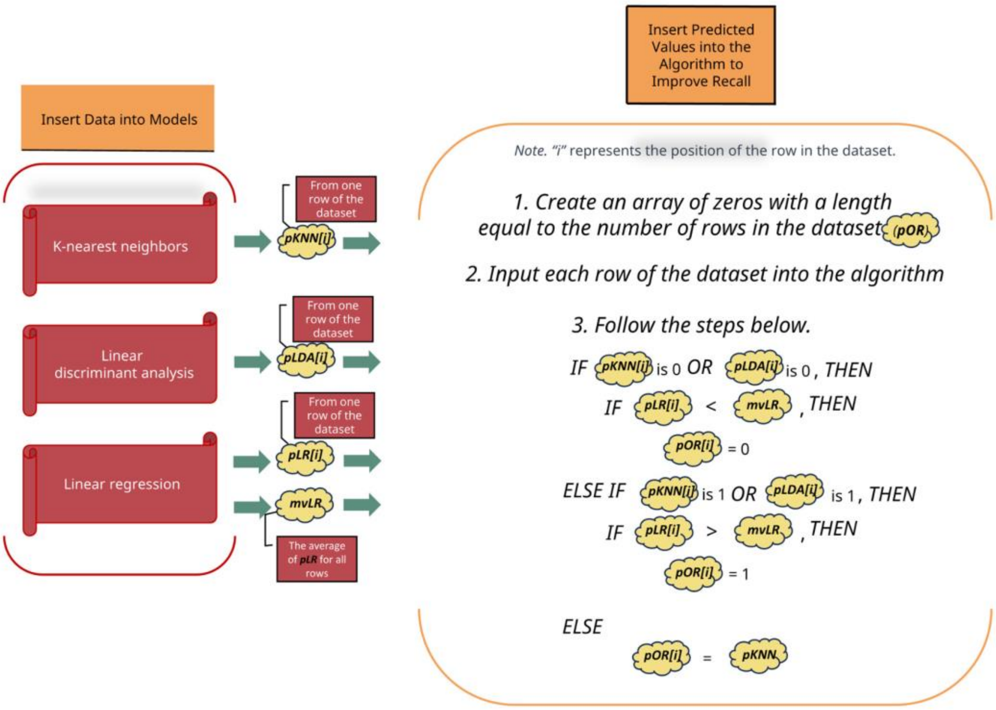
Input:

pKNN = A predicted value from KNN pLDA = A predicted value from LDA pLR = A predicted value from LR mvLR = A mean value from LR

Output:

pOR = Predicted value from our methodology
FOR i FROM 0 to array of zeros with a length of a dataset DO

/*If “non-fraud” Comes Out from Both Models*/
IF (pKNN[i] is 0 OR pLDA[i] is 0) THEN
IF (pLR[i] < mvLR) THEN pOR[i] ← 0
END IF
/*If “fraud” Comes Out from Both Models*/ ELSE IF (pKNN[i] is 1 OR pLDA[i] is 1) THEN
IF (pLR[i] > mvLR) THEN pOR[i] ← 1
END IF
/*Allocating Predicted Values from KNN to Remainings*/ ELSE
pOR [i] ← pKNN[i] END IF
END FOR


Insert Data into Models
K-nearest neighbors
Linear discriminant analysis
Linear regression
From one row of the dataset
pKNN[i]
From one row of the dataset
pLDA[i]
From one row of the dataset
pLR[i]
mvLR
The average of pLR for all rows
Insert Predicted Values into the Algorithm to Improve Recall
Note. "i" represents the position of the row in the dataset.
1. Create an array of zeros with a length equal to the number of rows in the dataset (pOR)
2. Input each row of the dataset into the algorithm
3. Follow the steps below.
IF pKNN[i] is 0 OR pLDA[i] is 0 , THEN
IF pLR[i] < mvLR , THEN
pOR[i] = 0
ELSE IF pKNN[i] is 1 OR pLDA[i] is 1 , THEN
IF pLR[i] > mvLR , THEN
pOR[i] = 1
ELSE
pOR[i] = pKNN

CHAPTER 2

MERITS AND DEMERITS OF THE BASE PAPER

LITERATURE SURVEY:

There are various algorithms available to detect fraud in financial transactions. Some of them are listed below mentioning the merits and demerits of the proposed method over each of the existing methods.

- **"Transaction Fraud Detection Based on Total Order Relation and Behaviour Diversity"**
  By Lutao Zheng. This paper extracts the behaviour profile of the user and verify the incoming transaction in the view of behaviour profile. Logical graph of BP(LGBP) is used to find logical relationship between the attributes. OM(Model proposed in paper) overcomes the shortcoming of Markov chain models since it characterizes the diversity of user behaviors . But the main disadvantage of model is that for high stability group (HS) SM method is better than ours(OM) since SM is based on Markov chain, and thus is more suitable for the stable case

- **Improved competitive learning neural networks for network intrusion and fraud detection** By John Zhong Lei. This paper proposes 2 new clustering algorithm, the improved competitive learning network (ICLN) and the supervised improved competitive learning network (SICLN), for fraud detection. This achieves low misclassification rate in solving classification problems and is able to deal with both labeled and unlabeled data. The main disadvantage of this model is they can't guarantee avoiding local optimisation

- **Teaching the Basics of KNN, LDA and Simple Perceptron Algorithms for Binary Classification Problems** by Lopez-Bernal. This paper uses 3 algorithms , K-Nearest-Neighbor (KNN), Linear Discriminant Analysis (LDA), Simple Perceptron. The main Advantage of using KNN is it's fast training and is easy to understand. LDA has a very low computation part cost and easy to implement. Perceptron is easy to train and setup. Main disadvantage of KNN is it's high computation cost and poor run time performance. LDA requires normal distribution and limited to 2 classes. Perceptron only works on linearly seperable data and limited to binary data

- **A novel idea for credit card fraud detection using decision tree** by Tiwarekar. This paper proposes a system which uses decision tree with Luhn's and Hunt's Algirthm to detect fraud transaction. Computational cost to run the proposed framework on large dataset is considerably high. Since different algorithms are used complex to understand and trouble shoot the problem

MERITS AND DEMERITS

Merits:

- Improved accuracy: By combining multiple methods, the system may be able to capture different aspects of fraud that a single method might miss. This could potentially lead to more accurate fraud detection .

- Reduced False Positives: By combining multiple models, the system can potentially flag fewer legitimate transactions as fraudulent. This reduces the hassle for customers whose cards are mistakenly blocked and improves overall customer experience .

- Potential for Scalability: This system can potentially be scaled to handle larger datasets efficiently. This is important as the volume of credit card transactions continues to grow. By distributing the workload across different models, the system can maintain good performance on larger datasets .

Demerits:

- Debugging Challenges: Troubleshooting issues in a multi-model system can be time-consuming. Isolating the source of an error within a specific model or the method for combining outputs can be difficult.

- Potential for Cascading Errors: Errors in one model can propagate through the system, impacting the overall accuracy of fraud detection. Robust error handling mechanisms become essential to mitigate this risk.

- Complexity: The integration of multiple models and conditions may increase the complexity of the algorithm, potentially impacting its efficiency and interpretability.

- Time: This system takes long time to train since many models are integrated with each other .

CHAPTER 3
SOURCE CODE

3.1 Dataset-1

Dataset1.py

```
import pandas as pd
import numpy as np
dataset=pd.read_csv('PS_20174392719_1491204439457_log.csv')
missing=dataset.isnull().sum()
print(missing)
dataset.drop(labels=['oldbalanceOrg','newbalanceOrig','oldbalanceDest','newbalanceDest']
,axis=1)
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
dataset['type'] = label_encoder.fit_transform(dataset['type'])
print("stage {}".format(1))

dataset['nameOrig'] = label_encoder.fit_transform(dataset['nameOrig'])
print("stage {}".format(2))

dataset['nameDest'] = label_encoder.fit_transform(dataset['nameDest'])
print("stage {}".format(3))

dataset.to_csv("preprocessed_data.csv", index=False)
print("stage {}".format(4))
dataset=pd.read_csv('preprocessed_data.csv')
missing=dataset.isnull().sum()
dataset=pd.concat([dataset,pd.get_dummies(dataset['type'], prefix='type_')],axis=1)
dataset.drop(['type'],axis=1,inplace = True)

dataset.head()
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()

dataset['type__0'] = label_encoder.fit_transform(dataset['type__0'])
print("stage {}".format(1))

dataset['type__1'] = label_encoder.fit_transform(dataset['type__1'])
print("stage {}".format(2))
```

```
dataset['type__2'] = label_encoder.fit_transform(dataset['type__2'])
print("stage {}".format(2))

dataset['type__3'] = label_encoder.fit_transform(dataset['type__3'])
print("stage {}".format(3))

dataset['type__4'] = label_encoder.fit_transform(dataset['type__4'])
print("stage {}".format(4))

dataset.to_csv("preprocessed_data.csv", index=False)
print("stage {}".format(5))
x=dataset.drop(columns=['isFlaggedFraud','isFraud'])
y=dataset['isFraud']
from imblearn.over_sampling import SMOTE
smote=SMOTE(sampling_strategy='minority')
x_sm,y_sm=smote.fit_resample(x,y)
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x_sm,y_sm,test_size=.2,stratify=y_sm)
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
knn = KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=-1, n_neighbors=5, p=2, weights='uniform')
knn.fit(xtrain,ytrain)
ypre_knn=knn.predict(xtest)
print("report\n",classification_report(ytest,ypre_knn,digits=6))
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score,
precision_score, f1_score, classification_report

lda = LinearDiscriminantAnalysis(covariance_estimator=None, n_components=None,
priors=None, shrinkage=None, solver='svd', store_covariance=False, tol=0.0001)
lda.fit(xtrain, ytrain)
ypre_lda = lda.predict(xtest)
print("lda report\n", classification_report(ytest, ypre_lda))
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(xtrain,ytrain)
ypre_lr=lr.predict(xtest)

mse = mean_squared_error(ytest, ypre_lr)
mae = mean_absolute_error(ytest, ypre_lr)
rmse = np.sqrt(mse)
```

```
print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R2) Score:", rmse)
mvl=sum(ypre_lr)/len(ypre_lr)
por=[0]*len(ytest)
for i in range(0,len(ypre_lr)):
    if(ypre_knn[i]==0 or ypre_lda[i]==0):
        if(ypre_lr[i]<mvl):
            por[i]=0
    elif(ypre_knn[i]==1 or ypre_lda[i]==1):
        if(ypre_lr[i]>mvl):
            por[i]=1
    else:
        por[i]=ypre_knn[i]
acc=accuracy_score(ytest,por)
pcc=precision_score(ytest,por)
ff=f1_score(ytest,por)
re=recall_score(ytest,por)
print("acc : ",acc)
print("pre : ",pcc)
print("f1  : ",ff)
print("re  : ",re)
print(classification_report(ytest,por))
from sklearn import tree
clf = tree.DecisionTreeClassifier()
clf = clf.fit(xtrain,ytrain)
ydt=clf.predict(xtest)

print("lda report\n", classification_report(ytest, ydt))
acc=accuracy_score(ytest,ydt)
pcc=precision_score(ytest,ydt)
ff=f1_score(ytest,ydt)
re=recall_score(ytest,ydt)
print("acc : ",acc)
print("pre : ",pcc)
print("f1  : ",ff)
print("re  : ",re)
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
clf = RandomForestClassifier(max_depth=2, random_state=0)
clf.fit(xtrain, ytrain)
yrf=clf.predict(xtest)
```

```
print("lda report\n", classification_report(ytest, yrf))
acc=accuracy_score(ytest,yrf)
pcc=precision_score(ytest,yrf)
ff=f1_score(ytest,yrf)
re=recall_score(ytest,yrf)
print("acc : ",acc)
print("pre : ",pcc)
print("f1  : ",ff)
print("re  : ",re)
from sklearn.ensemble import ExtraTreesClassifier
clf = ExtraTreesClassifier(n_estimators=100, random_state=0)
clf.fit(xtrain, ytrain)
yet=clf.predict(xtest)
print("lda report\n", classification_report(ytest, yet))
acc=accuracy_score(ytest,yet)
pcc=precision_score(ytest,yet)
ff=f1_score(ytest,yet)
re=recall_score(ytest,yet)
print("acc : ",acc)
print("pre : ",pcc)
print("f1  : ",ff)
print("re  : ",re)
from sklearn.ensemble import AdaBoostClassifier
clf = AdaBoostClassifier(n_estimators=100, algorithm="SAMME", random_state=0)
clf.fit(xtrain, ytrain)
yab=clf.predict(xtest)

print("lda report\n", classification_report(ytest, yab))
acc=accuracy_score(ytest,yab)
pcc=precision_score(ytest,yab)
ff=f1_score(ytest,yab)
re=recall_score(ytest,yab)
print("acc : ",acc)
print("pre : ",pcc)
print("f1  : ",ff)
print("re  : ",re)
```

3.2 Dataset-2

Dataset2.py

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
import warnings
df = pd.read_csv('fraudTrain.csv')
df1=pd.read_csv('fraudTest.csv')
ypre_KNN = pd.read_csv('krish_ypred_knn.csv')
label_encoder = LabelEncoder()
df['encoded_merchant'] = label_encoder.fit_transform(df['merchant'])
df['encoded_trans_date_trans_time'] = label_encoder.fit_transform(df['trans_date_trans_time'])
df['encoded_category'] = label_encoder.fit_transform(df['category'])
df['encoded_first'] = label_encoder.fit_transform(df['first'])
df['encoded_last'] = label_encoder.fit_transform(df['last'])
df['encoded_gender'] = label_encoder.fit_transform(df['gender'])
df['encoded_street'] = label_encoder.fit_transform(df['street'])
df['encoded_job'] = label_encoder.fit_transform(df['job'])
df['encoded_transNum'] = label_encoder.fit_transform(df['trans_num'])
df['encoded_city'] = label_encoder.fit_transform(df['city'])
df['encoded_state'] = label_encoder.fit_transform(df['state'])
df['encoded_dob'] = label_encoder.fit_transform(df['dob'])
label_encoder = LabelEncoder()
df1['encoded_merchant'] = label_encoder.fit_transform(df1['merchant'])
df1['encoded_trans_date_trans_time'] = label_encoder.fit_transform(df1['trans_date_trans_time'])
df1['encoded_category'] = label_encoder.fit_transform(df1['category'])
df1['encoded_first'] = label_encoder.fit_transform(df1['first'])
df1['encoded_last'] = label_encoder.fit_transform(df1['last'])
df1['encoded_gender'] = label_encoder.fit_transform(df1['gender'])
df1['encoded_street'] = label_encoder.fit_transform(df1['street'])
df1['encoded_job'] = label_encoder.fit_transform(df1['job'])
df1['encoded_transNum'] = label_encoder.fit_transform(df1['trans_num'])
df1['encoded_city'] = label_encoder.fit_transform(df1['city'])
df1['encoded_state'] = label_encoder.fit_transform(df1['state'])
df1['encoded_dob'] = label_encoder.fit_transform(df1['dob'])
```

```
dataset=df.drop(['merchant','category','first','last','gender','street','job','trans_num','city','state','dob','
trans_date_trans_time'],axis=1)
dataset1=df1.drop(['merchant','category','first','last','gender','street','job','trans_num','city','state','do
b','trans_date_trans_time'],axis=1)
Y_train=dataset['is_fraud']
X_train = dataset.drop(['is_fraud'],axis=1)
X_test=dataset1.drop(['is_fraud'],axis=1)
Y_test=dataset1['is_fraud']
newy=pd.concat([Y_train,Y_test],axis=0)
newy.value_counts()
newx=pd.concat([X_train,X_test],axis=0)
smote=SMOTE(sampling_strategy='minority')
x_sm,y_sm=smote.fit_resample(newx,newy)
y_sm.value_counts()
xtrain,xtest,ytrain,ytest=train_test_split(x_sm,y_sm,test_size=0.2,stratify=y_sm)
knn=KNeighborsClassifier(algorithm='auto',leaf_size=30,metric='minkowski',metric_params=No
ne,n_jobs=-1,n_neighbors=5,p=2,weights='uniform')
knn.fit(xtrain,ytrain)
ypre_KNN=knn.predict(xtest)
cm=confusion_matrix(ytest,ypre_KNN)
print(classification_report(ytest,ypre_KNN))
print(cm)
to_store = pd.DataFrame({'Predicted Values': ypre_KNN})

# Saving the DataFrame to CSV
to_store.to_csv('krish_ypred_knn.csv', index=False)
lda =
LinearDiscriminantAnalysis(covariance_estimator=None,n_components=None,priors=None,shrin
kage=None,solver='svd',store_covariance=False,tol=0.0001)
lda.fit(xtrain, ytrain)
ypre_LDA = lda.predict(xtest)
cm=confusion_matrix(ytest,ypre_LDA)
print(classification_report(ytest,ypre_LDA))
lr=LinearRegression()
lr.fit(xtrain,ytrain)
ypre_lr=lr.predict(xtest)
mse = mean_squared_error(ytest, ypre_lr)
# rmse = mean_squared_error(ytest, ypre_lr, squared=False)
mae = mean_absolute_error(ytest, ypre_lr)
r2 = r2_score(ytest, ypre_lr)

print("Mean Squared Error (MSE):", mse)
# print("Root Mean Squared Error (RMSE):", rmse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R2) Score:", r2)

#algorithm
mvl=sum(ypre_lr)/len(ypre_lr)
```

```
por=[0]*len(ytest)
for i in range(len(ytest)):
    if(ypre_knn[i]==0 and ypre_LDA[i]==0):
            if(ypre_lr[i]<mvl):
                por[i]=0

    elif(ypre_knn[i]==1 and ypre_LDA[i]==1):
        if(ypre_lr[i]>mvl):
            por[i]=1
    else:
        por[i]=ypre_LDA[i]
from sklearn.metrics import classification_report
print(classification_report(ytest,por))
print(recall_score(ytest,por))
combined_predictions = []
for i in range(len(ypre_knn)):

    avg_proba = (ypre_knn[i] + ypre_LDA[i] + ypre_lr[i]) / 3.0


    if avg_proba > 0.43:
        combined_predictions.append(1)
    else:
        combined_predictions.append(0)
from sklearn.metrics import classification_report
print(classification_report(ytest,combined_predictions))
import seaborn as sns
import matplotlib.pyplot as plt
correlation_matrix = x_sm.corr()

# Plot the correlation matrix as a heatmap
plt.figure(figsize=(12, 12))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
# Select the numeric columns
numeric_columns = x_sm.select_dtypes(include=['int64', 'float64']).columns

# Plot histograms for each numeric column
for column in numeric_columns:
    plt.figure(figsize=(4, 4))
    plt.hist(x_sm[column], bins=30, color='skyblue', edgecolor='black')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.title(f'Distribution of {column}')
    plt.grid(True)
    plt.show()
```

3.3 Dataset-3

Dataset3.py

```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings("ignore")
df=pd.read_csv('credit_dataset.csv')
missing=df.isnull().sum()
print("Numbers of Rows : ", df.shape[0])
print("Number of columns : ", df.shape[1])
print(df.info())
df.head()
#Dropping the the FLAG_MOBIL attribute as it has only 1 unique value

df.drop(columns=['FLAG_MOBIL'], inplace=True)
fraud_trans = df['TARGET'].value_counts()[1]
valid_trans = df['TARGET'].value_counts()[0]

print("Number of Fraudulent Transaction : ",fraud_trans," =" ,(fraud_trans/25134)*100)
print("Number of Valid Transaction : " ,valid_trans,"=",(valid_trans/25134)*100)
#Encoding the Attributes :
'''
gender”, “car”, “reality”,
“income_type”, “education_type”, “house_type”, and “family_type '''

print("Gender : ",df['GENDER'].unique())
print("car : ",df['CAR'].unique())
print("reality : ",df['REALITY'].unique())
print("income_type : ",df['INCOME_TYPE'].unique())
print("education_type : ",df['EDUCATION_TYPE'].unique())
print("house_type : ",df['HOUSE_TYPE'].unique())
print("family_type : ",df['FAMILY_TYPE'].unique())
from sklearn.preprocessing import LabelEncoder

columns_to_encode = ["GENDER", "CAR", "REALITY", "FAMILY_TYPE",
"INCOME_TYPE", "EDUCATION_TYPE", "HOUSE_TYPE"]
le = LabelEncoder()

for column in columns_to_encode:
    encoded = le.fit_transform(df[column])
    df[column] = encoded

df.head()
df['INCOME'] = df['INCOME'].astype('int64')
df['FAMILY SIZE'] = df['FAMILY SIZE'].astype('int64')
X=df.drop(columns='TARGET')
```

```
y=df['TARGET']

print(X.info())
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size= 0.25, random_state=0)
from sklearn.tree import DecisionTreeClassifier

# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
 from sklearn.metrics import recall_score

# Calculating recall
recall = recall_score(y_test, y_pred)
recall
from sklearn.metrics import classification_report

report = classification_report(y_test, y_pred)
print(report)
# Import necessary libraries and functions
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report


# Create Random Forest classifier object
rf_clf = RandomForestClassifier(random_state=0)

# Train the Random Forest classifier
rf_clf.fit(X_train, y_train)

# Predict the response for the test dataset
y_pred = rf_clf.predict(X_test)

# Print the classification report
print(classification_report(y_test, y_pred))
from imblearn import under_sampling, over_sampling
from imblearn.over_sampling import SMOTE

smote=SMOTE(sampling_strategy='minority')
x_sm,y_sm=smote.fit_resample(X,y)
#Target , Age , Family_Size , Phone , No_of_Child , Unnamed
```

```
#Correlation matrix

import seaborn as sns
import matplotlib.pyplot as plt

# Select the columns you want to include in the correlation matrix
columns = ['TARGET', 'AGE', 'FAMILY SIZE', 'PHONE', 'NO_OF_CHILD']

# Compute the correlation matrix
corr_matrix = df[columns].corr()

# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")  # fmt=".2f" for 2 decimal
places
plt.title('Correlation Matrix')
plt.show()
column_names = list(df.columns)

column_names.remove("TARGET")

print(column_names)
print(len(column_names))
from sklearn.datasets import make_regression
from xgboost import XGBRegressor
from matplotlib import pyplot

# Define dataset
X, y = make_regression(n_samples=1000, n_features=19, n_informative=5, random_state=1)

# Define the model
model = XGBRegressor()

# Fit the model
model.fit(X, y)

# Get importance
importance = model.feature_importances_

# Get feature names from the dataset (assuming you have them)
feature_names = ['Unnamed: 0', 'ID', 'GENDER', 'CAR', 'REALITY', 'NO_OF_CHILD',
'INCOME', 'INCOME_TYPE', 'EDUCATION_TYPE', 'FAMILY_TYPE', 'HOUSE_TYPE',
'FLAG_MOBIL', 'WORK_PHONE', 'PHONE', 'E_MAIL', 'FAMILY SIZE', 'BEGIN_MONTH',
'AGE', 'YEARS_EMPLOYED'] # Modify this based on your actual feature name
# Summarize feature importance
for i,v in enumerate(importance):
    print(f'{feature_names[i]}: {v:.5f}')
```

```
# Plot feature importance
fig, ax = pyplot.subplots(figsize=(10, 6))  # Adjust figure size as needed
bar = ax.bar(feature_names, importance)
ax.set_xlabel('Features')
ax.set_ylabel('Importance Score')
ax.set_title('Feature Importance')
ax.tick_params(axis='x', rotation=45)  # Rotate x-axis labels for better readability
pyplot.show()

from sklearn.neighbors import KNeighborsClassifier
 from sklearn.model_selection import StratifiedKFold
 from sklearn.metrics import recall_score
 from sklearn.linear_model import LinearRegression
 from sklearn.metrics import mean_squared_error, mean_absolute_error
 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
 import numpy as np

 # Define the top N significant attributes
 significant_attributes = {
     'Unnamed: 0': 0.13839,
     'ID': 0.00104,
     'GENDER': 0.00154,
     'CAR': 0.00199,
     'REALITY': 0.00109,
     'NO_OF_CHILD': 0.30016,
     'INCOME': 0.00058,
     'INCOME_TYPE': 0.00115,
     'EDUCATION_TYPE': 0.53974,
     'FAMILY_TYPE': 0.00191,
     'HOUSE_TYPE': 0.00220,
     'WORK_PHONE': 0.00224,
     'PHONE': 0.00072,
     'E_MAIL': 0.00280,
     'FAMILY SIZE': 0.00092,
     'BEGIN_MONTH': 0.00080,
     'AGE': 0.00081,
     'YEARS_EMPLOYED': 0.00089
 }
 # Sort the attributes based on their importance
 sorted_attributes = sorted(significant_attributes.items(), key=lambda x: x[1], reverse=True)
 # Define lists to store recall values for kNN, LDA, and LR
 recall_knn_list = []
 recall_lda_list = []
 mse_LR = []
 mae_LR = []
 rmse_LR = []

 pKNN,pLDA,pLR =[],[],[]
```

```
# Create StratifiedKFold object
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)

# Iterate over the range of top_n_attributes from 5 to 18
for top_n_attributes in range(5, 19):
    # Select the top N attributes
    top_n_attributes_names = [attr for attr, _ in sorted_attributes[:top_n_attributes]]

    # Initialize lists to store recall values for each fold
    recall_knn_fold = []
    recall_lda_fold = []
    recall_lr_fold = []

    # Iterate over the folds
    for train_index, test_index in skf.split(x_sm, y_sm):
        x_train_fold, x_test_fold = x_sm[top_n_attributes_names].iloc[train_index],
x_sm[top_n_attributes_names].iloc[test_index]
        y_train_fold, y_test_fold = y_sm.iloc[train_index], y_sm.iloc[test_index]

        # Fit kNN model
        knn = KNeighborsClassifier()
        knn.fit(x_train_fold, y_train_fold)
        y_pred_knn = knn.predict(x_test_fold)
        pKNN.extend(y_pred_knn)
        recall_knn = recall_score(y_test_fold, y_pred_knn)
        recall_knn_fold.append(recall_knn)

        # Fit LDA model
        lda = LinearDiscriminantAnalysis()
        lda.fit(x_train_fold, y_train_fold)
        y_pred_lda = lda.predict(x_test_fold)
        pLDA.extend(y_pred_lda)
        recall_lda = recall_score(y_test_fold, y_pred_lda)
        recall_lda_fold.append(recall_lda)

        # Fit LR model
        LR = LinearRegression()
        LR.fit(x_train_fold, y_train_fold)
        y_LR = LR.predict(x_test_fold)
        pLR.extend(y_LR)
        mse = mean_squared_error(y_test_fold, y_LR)
        mae = mean_absolute_error(y_test_fold, y_LR)
        rmse = np.sqrt(mse)

        mse_LR.append(mse)
        mae_LR.append(mae)
        rmse_LR.append(rmse)
```

```
    # Calculate the mean recall values for each model and append to the respective lists
    recall_knn_list.append(np.mean(recall_knn_fold))
    recall_lda_list.append(np.mean(recall_lda_fold))
     recall_lr_list.append(np.mean(recall_lr_fold))

 # Print the recall values for each top_n_attributes
 for i, top_n_attributes in enumerate(range(5, 19)):
    print(f"Top {top_n_attributes} Attributes:")
    print(f"Mean Recall (kNN): {recall_knn_list[i]}")
    print(f"Mean Recall (LDA): {recall_lda_list[i]}")
    print(f"Mean Recall (LR): {recall_lr_list[i]}")
    print()
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x_sm,y_sm,test_size=.20)
#kNN

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
knn = KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=-1, n_neighbors=5, p=2, weights='uniform')

knn.fit(xtrain,ytrain)
ypre_knn=knn.predict(xtest)
print("report\n",classification_report(ytest,ypre_knn,digits=6))
import matplotlib.pyplot as plt

# Count the occurrences of 0s and 1s in ypre_knn
counts = [len(ypre_knn[ypre_knn == 0]), len(ypre_knn[ypre_knn == 1])]

# Define labels for the pie chart
labels = ['0 Prediction', '1 Prediction']

# Plot the pie chart
plt.figure(figsize=(8, 8))
plt.pie(counts, labels=labels, autopct='%1.1f%%', startangle=140)
plt.title('Distribution of Predictions')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
#LDA

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, precision_score,
f1_score, classification_report

lda = LinearDiscriminantAnalysis(covariance_estimator=None, n_components=None,
priors=None, shrinkage=None, solver='svd', store_covariance=False, tol=0.0001)
```

```
lda.fit(xtrain, ytrain)
ypre_lda = lda.predict(xtest)
print("lda report\n", classification_report(ytest, ypre_lda))
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
from sklearn.linear_model import LinearRegression

lr=LinearRegression()
lr.fit(xtrain,ytrain)
ypre_lr=lr.predict(xtest)

mse = mean_squared_error(ytest, ypre_lr)
mae = mean_absolute_error(ytest, ypre_lr)


print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
# mvl=sum(ypre_lr)/len(ypre_lr)
# por=[0]*len(ytest)

# Calculate the mean prediction of LR model
mvl = np.mean(ypre_lr)

# Initialize the final predictions array
por = np.zeros_like(ytest)

for i in range(0,len(ypre_lr)):
    if(ypre_knn[i]==0 or ypre_lda[i]==0):
        if(ypre_lr[i]<mvl):
            por[i]=0
    elif(ypre_knn[i]==1 or ypre_lda[i]==1):
        if(ypre_lr[i]>mvl):
            por[i]=1
    else:
        por[i]=ypre_knn[i]
print(classification_report(ytest,por))
import numpy as np

# Calculate the mean prediction of LR model
mvl = np.mean(ypre_lr)

# Initialize the final predictions array
por = np.zeros_like(ytest)

# Iterate over predictions to determine final predictions
for i in range(len(ypre_lr)):
    # Simplify conditions for clarity
    if (ypre_knn[i] == 0 or ypre_lda[i] == 0) and ypre_lr[i] < mvl:
        por[i] = 0
```

```
    elif (ypre_knn[i] == 1 or ypre_lda[i] == 1) and ypre_lr[i] > mvl:
        por[i] = 1
    else:
        por[i] = ypre_knn[i]

print(classification_report(ytest,por))
acc=accuracy_score(ytest,por)
pcc=precision_score(ytest,por)
ff=f1_score(ytest,por)
re=recall_score(ytest,por)

print("acc : ",acc)
print("pre : ",pcc)
print("f1  : ",ff)
print("re  : ",re)
# Original attribute-value pairs
attributes = {
    'Unnamed: 0': 0.13839,
    'ID': 0.00104,
    'GENDER': 0.00154,
    'CAR': 0.00199,
    'REALITY': 0.00109,
    'NO_OF_CHILD': 0.30016,
    'INCOME': 0.00058,
    'INCOME_TYPE': 0.00115,
    'EDUCATION_TYPE': 0.53974,
    'FAMILY_TYPE': 0.00191,
    'HOUSE_TYPE': 0.00220,
    'FLAG_MOBIL': 0.00102,
    'WORK_PHONE': 0.00224,
    'PHONE': 0.00072,
    'E_MAIL': 0.00280,
    'FAMILY SIZE': 0.00092,
    'BEGIN_MONTH': 0.00080,
    'AGE': 0.00081,
    'YEARS_EMPLOYED': 0.00089
}

# Sort the attributes in descending order of values
sorted_attributes = sorted(attributes.items(), key=lambda x: x[1], reverse=True)

# Print the sorted attributes
for attr, value in sorted_attributes:
    print(f"{attr}: {value}")
import numpy as np

# Inputs: Predicted values from different models
PKNN = ypre_knn
```

```
PLDA = ypre_lda
PLR = ypre_lr

# Output: Predicted value from the algorithm
por = np.zeros_like(ytest)

# Define the threshold
threshold = 0.43

# Calculate POR according to the algorithm
for i in range(len(ytest)):
    # Calculate the average of the predicted values from KNN, LDA, and LR
    avg = (PKNN[i] + PLDA[i] + PLR[i]) / 3

    # Compare the average to the threshold
    if avg > threshold:
        por[i] = 1
    else:
        por[i] = 0

# Now 'por' contains the final predictions according to the algorithm
print(classification_report(ytest,por))
import numpy as np

# Inputs: Predicted values from different models
PKNN = ypre_knn
PLDA = ypre_lda
PLR = ypre_lr

# Output: Predicted value from the algorithm
por = np.zeros_like(ytest)

# Define the threshold
threshold = 0.5

# Calculate POR according to the algorithm
for i in range(len(ytest)):
    # Calculate the average of the predicted values from KNN, LDA, and LR
    avg = (PKNN[i] + PLDA[i] + PLR[i]) / 3

    # Compare the average to the threshold
    if avg > threshold:
        por[i] = 1
    else:
        por[i] = 0

# Now 'por' contains the final predictions according to the algorithm
print(classification_report(ytest,por))
```

3.4 Dataset-4

Dataset4.py

```
import pandas as pd
df = pd.read_csv('train_transaction.csv')
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# List of columns to apply label encoding
columns_to_encode = ['ProductCD', 'card1', 'card2', 'card3', 'card4', 'card5', 'card6',
                 'addr1', 'addr2', 'P_emaildomain', 'R_emaildomain', 'M1', 'M2', 'M3', 'M4', 'M5', 'M6',
'M7', 'M8', 'M9']

# Apply label encoding for each column
for column in columns_to_encode:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column].astype(str))

# Calculate the number of missing values per column
missing_values_per_column = df.isnull().sum()

# Calculate the median of missing values
median_missing_values = missing_values_per_column.median()

# Exclude columns with missing values surpassing the median
columns_to_exclude = missing_values_per_column[missing_values_per_column >
median_missing_values].index
df_cleaned = df.drop(columns=columns_to_exclude)

# Impute missing values using median of respective columns
df_cleaned.fillna(df_cleaned.median(), inplace=True)

fraud_proportion = df['isFraud'].mean()

print("Proportion of fraud cases in the dataset:", fraud_proportion)

# Now df_cleaned contains the preprocessed dataset with label encoded categorical columns and
missing values imputed
x=df_cleaned.drop(columns=['isFraud'])
y=df_cleaned['isFraud']
print(len(df_cleaned))
print(len(x),len(y))
from imblearn.over_sampling import SMOTE
smote=SMOTE(sampling_strategy='minority')
x_sm,y_sm=smote.fit_resample(x,y)
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x_sm,y_sm,test_size=.2,stratify=y_sm)
```

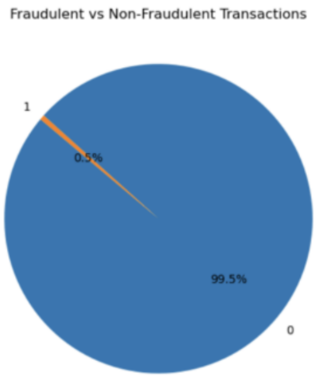
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
knn = KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=-1, n_neighbors=5, p=2, weights='uniform')
knn.fit(xtrain,ytrain)
ypre_knn=knn.predict(xtest)
print("Knn report\n",classification_report(ytest,ypre_knn,digits=6))
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, precision_score,
f1_score, classification_report

lda = LinearDiscriminantAnalysis(covariance_estimator=None, n_components=None,
priors=None, shrinkage=None, solver='svd', store_covariance=False, tol=0.0001)
lda.fit(xtrain, ytrain)
ypre_lda = lda.predict(xtest)
print("lda report\n", classification_report(ytest, ypre_lda))
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(xtrain,ytrain)
ypre_lr=lr.predict(xtest)

mse = mean_squared_error(ytest, ypre_lr)
mae = mean_absolute_error(ytest, ypre_lr)


print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
mvl=sum(ypre_lr)/len(ypre_lr)
por=[0]*len(ytest)
for i in range(0,len(ypre_lr)):
    if(ypre_knn[i]==0 or ypre_lda[i]==0):
        if(ypre_lr[i]<mvl):
            por[i]=0
    elif(ypre_knn[i]==1 or ypre_lda[i]==1):
        if(ypre_lr[i]>mvl):
            por[i]=1
    else:
        por[i]=ypre_knn[i]
acc=accuracy_score(ytest,por)
pcc=precision_score(ytest,por)
ff=f1_score(ytest,por)
re=recall_score(ytest,por)
print("acc : ",acc)
print("pre : ",pcc)
print("f1  : ",ff)
print("re  : ",re)
print(classification_report(ytest,por))
```

```
from sklearn import tree
clf = tree.DecisionTreeClassifier()
clf = clf.fit(xtrain,ytrain)
ydt=clf.predict(xtest)

print("DT report\n", classification_report(ytest, ydt))
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
clf = RandomForestClassifier(max_depth=2, random_state=0)
clf.fit(xtrain, ytrain)
yrf=clf.predict(xtest)

print("RT report\n", classification_report(ytest, yrf))
from sklearn.ensemble import ExtraTreesClassifier
clf = ExtraTreesClassifier(n_estimators=100, random_state=0)
clf.fit(xtrain, ytrain)
yet=clf.predict(xtest)
print("ET report\n", classification_report(ytest, yet))
from sklearn.ensemble import AdaBoostClassifier
clf = AdaBoostClassifier(n_estimators=100, algorithm="SAMME", random_state=0)
clf.fit(xtrain, ytrain)
yab=clf.predict(xtest)

print("AB report\n", classification_report(ytest, yab))
```

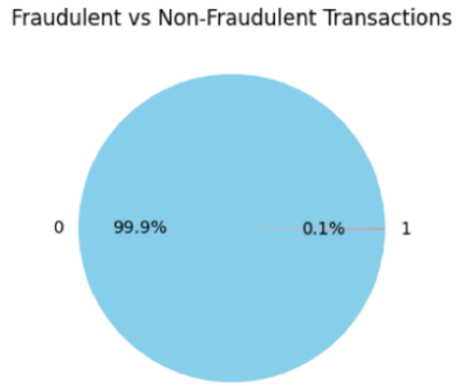
CHAPTER 4

OUTPUT SNAPSHOTS

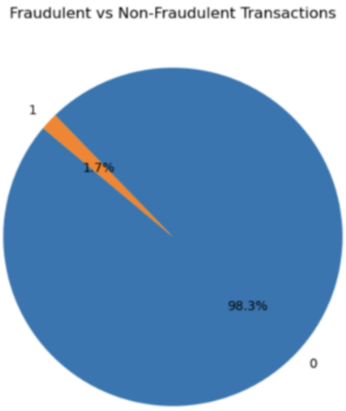
Dataset 1



Dataset 2



Dataset 3



Dataset 4

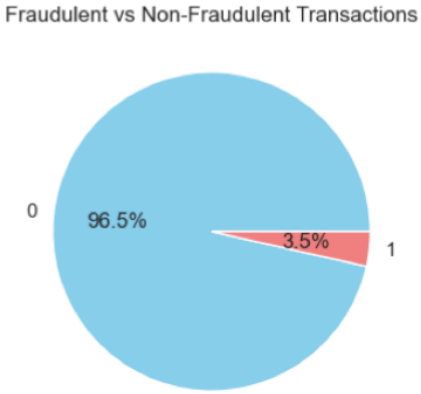
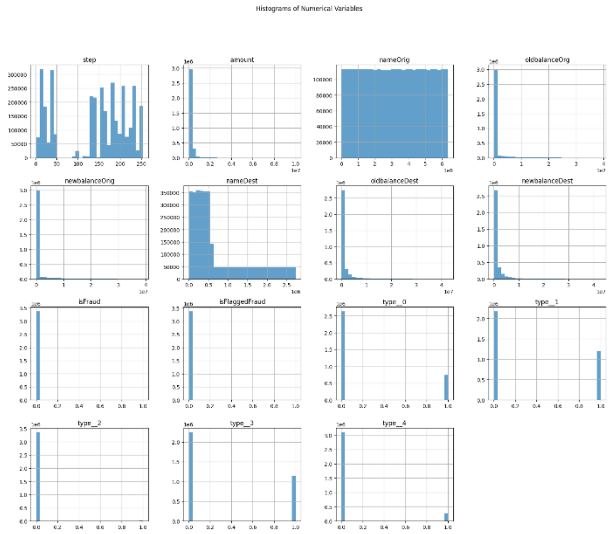
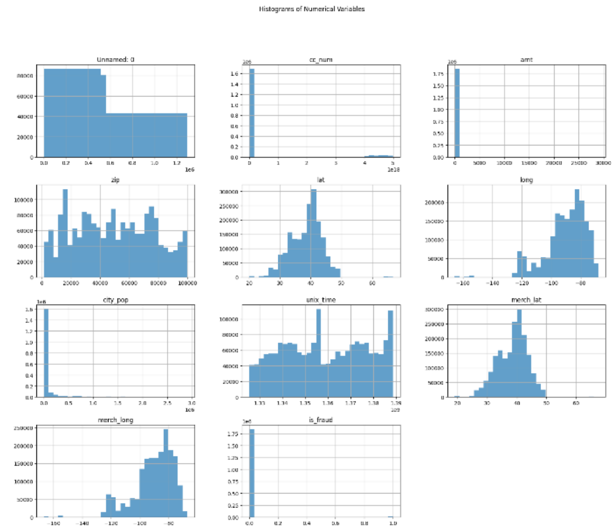


Fig 4.1. Distribution of 0 and 1 in the Target class using Pie Chart

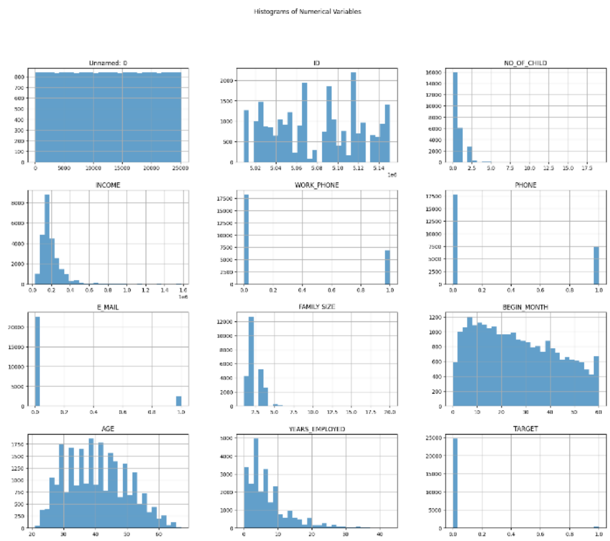
Dataset 1



Dataset 2



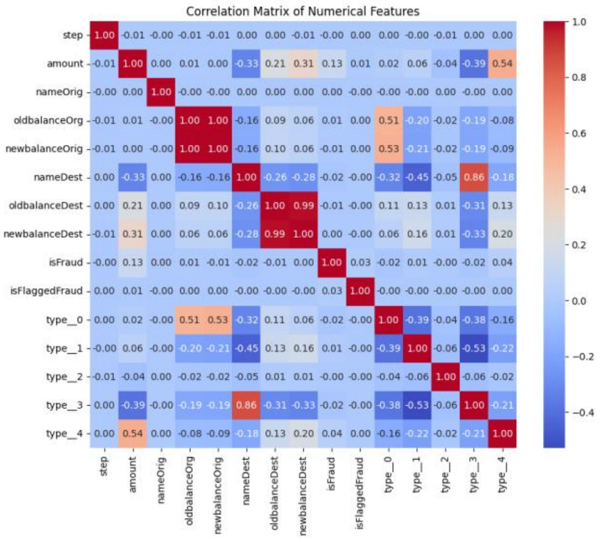
Dataset 3



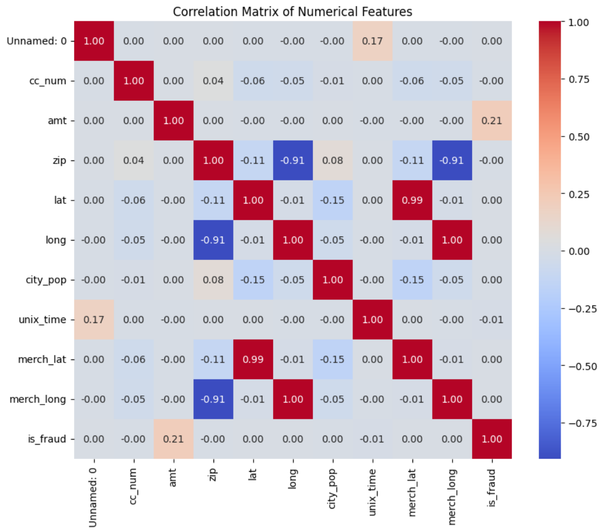
Dataset 4

Fig 4.2 Analyze the distribution of values in each attribute of the datasets

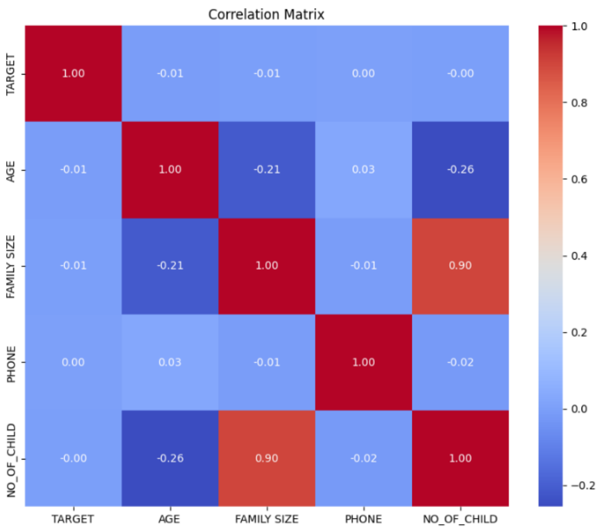
Dataset 1



Dataset 2



Dataset 3



Dataset 4

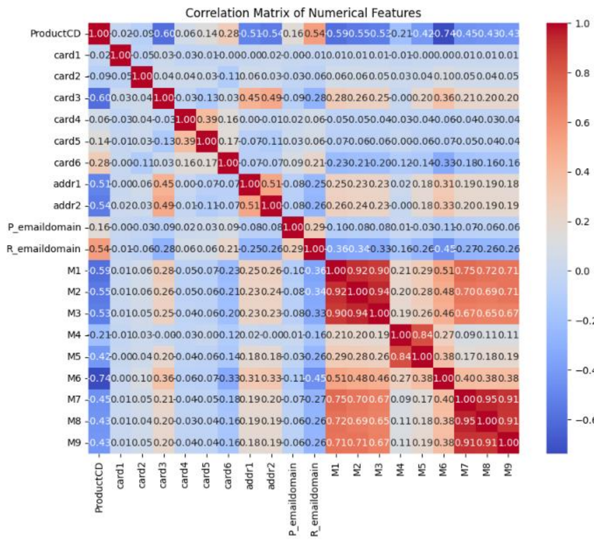
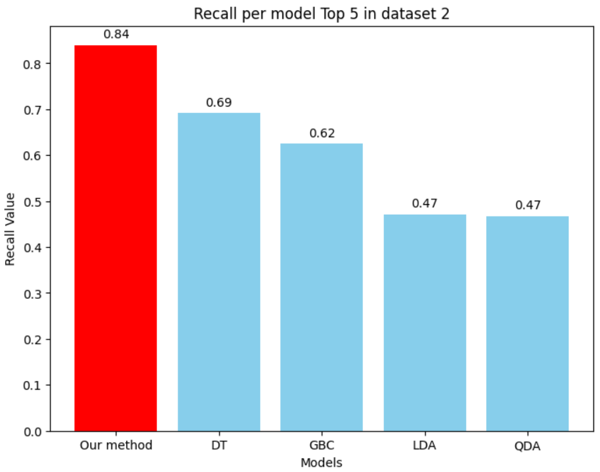


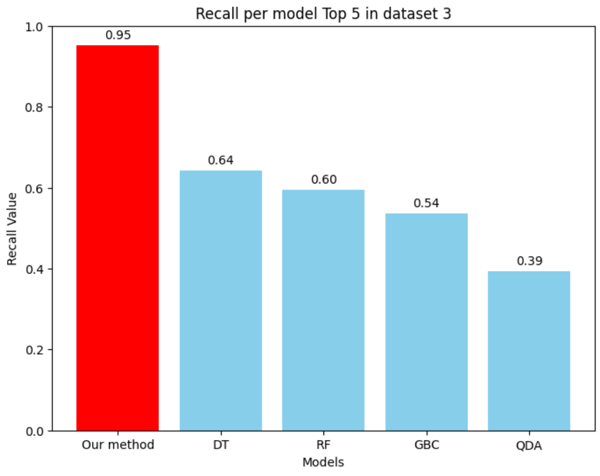
Fig 4.3 Correlation matrix

Dataset 1

Dataset 2



Dataset 3



Dataset 4

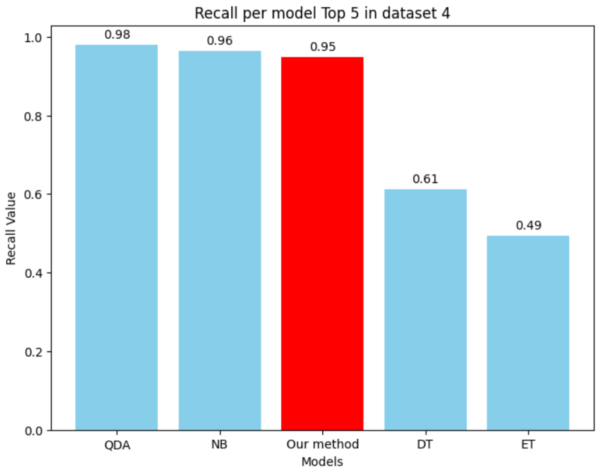


Fig 4.4 Comparison between the recall of our proposed methodology and other models

Dataset	Models	Base Paper	Our Implementation
1	Our method	100%	99%
1	DT	79%	80%
1	RF	78%	71%
1	ET	64%	62%
1	AB	57%	27%
2	Our method	97%	84%
2	DT	65%	69%
2	GBC	85%	62%
2	LDA	30%	47%
2	QDA	30%	47%
3	Our method	100%	95%
3	DT	64%	64%
3	RF	63%	60%
3	GBC	59%	39%
4	Our method	93%	95%
4	DT	56%	61%
4	ET	47%	49%
4	NB	95%	96%
4	QDA	98%	98%

Table 4.4 Comparison of Recall values of different models for different datasets

CHAPTER 5
CONCLUSION AND FUTURE PLANS

This Project proposes a methodology aimed at improving recall in credit card fraud detection across four distinct datasets. By preprocessing these datasets and prioritizing high recall while maintaining accuracy, this model yielded recall scores of 1.0000, 0.8301, 0.81, and 0.79 for the respective datasets.

In our proposed model we use KNN, LDA and LR to classify the values furthermore advanced machine learning techniques like Artificial Neural Network, Ensemble methods and Reinforcement learning can be used along with proposed algorithm for better recall.
This model is tested with finance transaction dataset only, in future it can be tested with healthcare dataset and other area.

CHAPTER 6

REFERENCES

1. **SMOTE Algorithm**:
    - https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/
    - https://towardsdatascience.com/upsampling-with-smote-for-classification-projects-e91d7c44e4bf

CHAPTER 7

APPENDIX

BASE PAPER

Jiwon Chung and KyunghoLee, "Credit Card Fraud Detection: An Improved Strategy for High Recall Using KNN, LDA, and Linear Regression," in Sensors 2023, 23, 7788.

doi: 10.3390/s23187788

keywords: {recall analysis; sensitivity analysis; true positive rate analysis; credit card fraud detection; KNN;LDA;linear regression},

URL: https://www.mdpi.com/1424-8220/23/18/7788