Configuration Manual

MSc Research Project
Data Analytics

Saran Raj Srinivasan
Student ID: x22149066

School of Computing
National College of Ireland

Supervisor: Noel Cosgrave

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Saran Raj Srinivasan
Student ID:	x22149066
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Noel Cosgrave
Submission Due Date:	31/01/2024
Project Title:	Configuration Manual
Word Count:	502
Page Count:	8

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Saran Raj Srinivasan
Date:	31st January 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Saran Raj Srinivasan
x22149066

1 Introduction

This document provides a detailed step by step information related to the environment needed to execute this research project "Comparative Analysis of Batch and Online Machine Learning Techniques for Fraud Detection: A Case Study on Real European Credit Card Transactions" . The complete project is performed using Python libraries in Jupyter notebook and Google Colab. This manual explain the system configuration used for the execution with the information about the data set along with tools and setups required to run the project.

2 System Specification

This section gives basic hardware and operating system requirements for this project:

2.1 Hardware

This project is executed with following hardware as default configuration on current laptop:

Laptop: HP Laptop 14s-fq1xxx
Processor: AMD Ryzen 3 5300U with Radeon Graphics, 2600 Mhz, 4 Core(s), 8 Logical Processor(s)
RAM: 24GB
Storage: 500GB

2.2 Software

The below software is configured on the current laptop used:

Operating System: Windows 11
Jupyter Notebook: Version 6.5.2
Python: Version 3.11.1

Google Colab Cloud: Used for executing Python code via the browser.
Runtime Type: Python 3
Hardware Accelerator: T4GPU

Python Package Upgrades: 'bigframes': Upgraded from 0.12.0 to 0.13.0
Python Package Inclusions: 'transformers': Version 4.35.2
'google-generativeai': Version 0.2.2

3 Development Environment

Python scripting language is used to develop this research project and below are the details of the environments utilized:

3.1 Anaconda

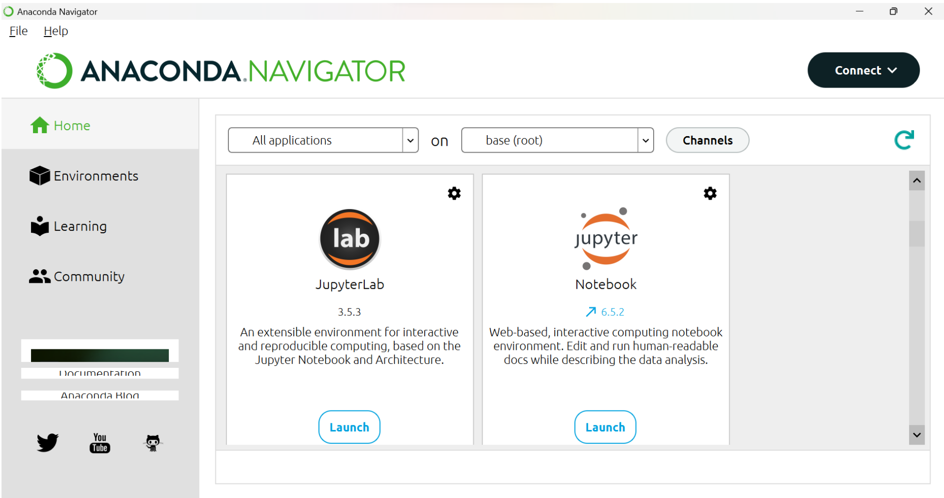

Figure 1: Anaconda

3.2 Jupyter notebook

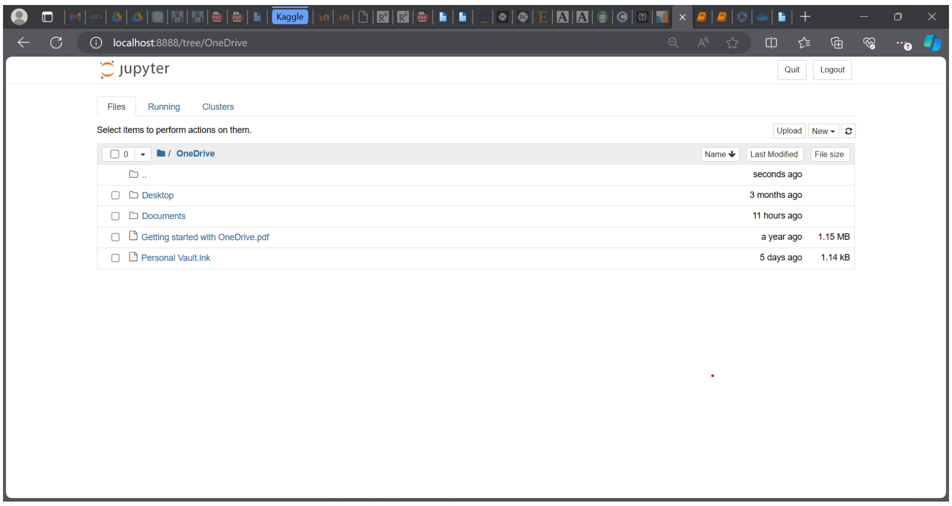

Figure 2: Jupyter Notebook

4 Implementation

4.1 Python Libraries

The following packages and libraries are utilized develop this research project, The pip command can be used to install new libraries if development requires them.
NumPy
Pandas
Matplotlib
Seaborn
Scikit-learn (Sklearn)
missingpy

```
In [1]: # Import necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.impute import KNNImputer
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from scipy.stats import spearmanr
from imblearn.over_sampling import RandomOverSampler
import sklearn.neighbors._base
import sys
sys.modules['sklearn.neighbors.base'] = sklearn.neighbors._base
import missingpy
```

Figure 3: Importing libraries and package

4.2 Dataset Details

The detailed instructions for executing the research project are provided below, along with a snapshot of the code.

Dataset - dataset_train_transaction

```
In [2]: #Reading the data from csv
dataset_train_transaction = pd.read_csv('RIC train_transaction.csv')
```

Figure 4: Loading the Data set

4.3 EDA, Preparation and Modelling

The below snapshots shows the details of Handling missing value, Imbalance data, Feature selection, Feature engineering, Modelling and Results.

```
In [6]: # print the column names
        print(list(dataset_train_transaction))

['TransactionID', 'isFraud', 'TransactionDT', 'TransactionAmt', 'ProductCD', 'card1', 'card2', 'card3', 'card4', 'card5', 'card
6', 'addr1', 'addr2', 'dist1', 'dist2', 'P_emaildomain', 'R_emaildomain', 'C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7', 'C8', 'C9',
'C10', 'C11', 'C12', 'C13', 'C14', 'D1', 'D2', 'D3', 'D4', 'D5', 'D6', 'D7', 'D8', 'D9', 'D10', 'D11', 'D12', 'D13', 'D14', 'D1
5', 'M1', 'M2', 'M3', 'M4', 'M5', 'M6', 'M7', 'M8', 'M9', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11',
'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'V29',
'V30', 'V31', 'V32', 'V33', 'V34', 'V35', 'V36', 'V37', 'V38', 'V39', 'V40', 'V41', 'V42', 'V43', 'V44', 'V45', 'V46', 'V47',
'V48', 'V49', 'V50', 'V51', 'V52', 'V53', 'V54', 'V55', 'V56', 'V57', 'V58', 'V59', 'V60', 'V61', 'V62', 'V63', 'V64', 'V65',
'V66', 'V67', 'V68', 'V69', 'V70', 'V71', 'V72', 'V73', 'V74', 'V75', 'V76', 'V77', 'V78', 'V79', 'V80', 'V81', 'V82', 'V83',
'V84', 'V85', 'V86', 'V87', 'V88', 'V89', 'V90', 'V91', 'V92', 'V93', 'V94', 'V95', 'V96', 'V97', 'V98', 'V99', 'V100', 'V101',
'V102', 'V103', 'V104', 'V105', 'V106', 'V107', 'V108', 'V109', 'V110', 'V111', 'V112', 'V113', 'V114', 'V115', 'V116', 'V117',
'V118', 'V119', 'V120', 'V121', 'V122', 'V123', 'V124', 'V125', 'V126', 'V127', 'V128', 'V129', 'V130', 'V131', 'V132', 'V133',
'V134', 'V135', 'V136', 'V137', 'V138', 'V139', 'V140', 'V141', 'V142', 'V143', 'V144', 'V145', 'V146', 'V147', 'V148', 'V149',
'V150', 'V151', 'V152', 'V153', 'V154', 'V155', 'V156', 'V157', 'V158', 'V159', 'V160', 'V161', 'V162', 'V163', 'V164', 'V165',
'V166', 'V167', 'V168', 'V169', 'V170', 'V171', 'V172', 'V173', 'V174', 'V175', 'V176', 'V177', 'V178', 'V179', 'V180', 'V181',
'V182', 'V183', 'V184', 'V185', 'V186', 'V187', 'V188', 'V189', 'V190', 'V191', 'V192', 'V193', 'V194', 'V195', 'V196', 'V197',
'V198', 'V199', 'V200', 'V201', 'V202', 'V203', 'V204', 'V205', 'V206', 'V207', 'V208', 'V209', 'V210', 'V211', 'V212', 'V213',
'V214', 'V215', 'V216', 'V217', 'V218', 'V219', 'V220', 'V221', 'V222', 'V223', 'V224', 'V225', 'V226', 'V227', 'V228', 'V229',
'V230', 'V231', 'V232', 'V233', 'V234', 'V235', 'V236', 'V237', 'V238', 'V239', 'V240', 'V241', 'V242', 'V243', 'V244', 'V245',
'V246', 'V247', 'V248', 'V249', 'V250', 'V251', 'V252', 'V253', 'V254', 'V255', 'V256', 'V257', 'V258', 'V259', 'V260', 'V261',
'V262', 'V263', 'V264', 'V265', 'V266', 'V267', 'V268', 'V269', 'V270', 'V271', 'V272', 'V273', 'V274', 'V275', 'V276', 'V277',
'V278', 'V279', 'V280', 'V281', 'V282', 'V283', 'V284', 'V285', 'V286', 'V287', 'V288', 'V289', 'V290', 'V291', 'V292', 'V293',
'V294', 'V295', 'V296', 'V297', 'V298', 'V299', 'V300', 'V301', 'V302', 'V303', 'V304', 'V305', 'V306', 'V307', 'V308', 'V309',
'V310', 'V311', 'V312', 'V313', 'V314', 'V315', 'V316', 'V317', 'V318', 'V319', 'V320', 'V321', 'V322', 'V323', 'V324', 'V325',
'V326', 'V327', 'V328', 'V329', 'V330', 'V331', 'V332', 'V333', 'V334', 'V335', 'V336', 'V337', 'V338', 'V339']
```

Figure 5: Missing values in Data set

```
In [22]: # Check for missing values in each column
         missing_values_transaction = dataset_train_transaction.isnull().sum()

         # Print columns with missing values and their respective counts
         for column, count in missing_values_transaction.items():
             if count > 0:
                 print(f"Column '{column}' has {count} missing values.")

         Column 'card2' has 8933 missing values.
         Column 'card3' has 1565 missing values.
         Column 'card4' has 1577 missing values.
         Column 'card5' has 4259 missing values.
         Column 'card6' has 1571 missing values.
         Column 'addr1' has 65706 missing values.
         Column 'addr2' has 65706 missing values.
         Column 'dist1' has 352271 missing values.
         Column 'dist2' has 552913 missing values.
         Column 'P_emaildomain' has 94456 missing values.
         Column 'R_emaildomain' has 453249 missing values.
         Column 'D1' has 1269 missing values.
         Column 'D2' has 280797 missing values.
         Column 'D3' has 262878 missing values.
         Column 'D4' has 168922 missing values.
         Column 'D5' has 309841 missing values.
         Column 'D6' has 517353 missing values.
         Column 'D7' has 551623 missing values.
         Column 'D8' has 515614 missing values.
```

Figure 6: Missing values in Data set

```
In [27]: # create a heatmap of missing values - dataset_train_identity
         sns.heatmap(dataset_train_identity.isnull(), cmap='viridis')
         plt.title('Missing Values Heatmap - dataset_train_identity')
         plt.show()
```

Figure 7: Heatmap to visualize the missing value in Data set

Number of missing values: 12

#['V279', 'V280', 'V284', 'V285', 'V286', 'V287', 'V290', 'V291', 'V292', 'V293', 'V294', 'V295', 'V297', 'V298', 'V299', 'V302', 'V303', 'V304', 'V305', 'V306', 'V307', 'V308', 'V309', 'V310', 'V311', 'V312', 'V316', 'V317', 'V318', 'V319', 'V320', 'V321']

```
In [30]: # Assuming 'group_columns' is a list of V columns for a specific group
         group_columns = ['V279', 'V280', 'V284', 'V285', 'V286', 'V287', 'V290', 'V291', 'V292', 'V293', 'V294', 'V295', 'V297', 'V298',
         # Create a DataFrame with only the group columns
         df_group = dataset_train_transaction[group_columns]

         # Compute the Spearman's correlation matrix
         corr_matrix = df_group.corr(method='spearman')

         # Plot the correlation matrix using a heatmap
         plt.figure(figsize=(20, 20))
         sns.heatmap(corr_matrix, cmap='coolwarm', annot=True)
         plt.show()
```

Figure 8: Wrapper subset method to group column with same missing value

Missing Pattern Check (MNAR) and Multiple Imputation

In [61]:

```
# Loading the first 1000 rows of the dataset
missing_data_first = dataset_train_transaction_new.head(20)

# Visualize missing data patterns
msno.matrix(missing_data_first)

# Performing Little's MCAR test
missing_data = missing_data_first.isnull()
missing_data = missing_data.sum()
missing_data = missing_data[missing_data > 0]
missing_data = missing_data.to_frame()
missing_data.columns = ['count']
missing_data['missing_percentage'] = (missing_data['count'] / len(missing_data_first)) * 100
```

Figure 9: Missing value Pattern- MNAR

In [7]:

```
# Initializing MICE imputer
mice_imputer = IterativeImputer()

# Performing multiple imputation on the subset
imputed_subset = mice_imputer.fit_transform(df)

# Creating a new DataFrame with the imputed values
imputed_df = pd.DataFrame(imputed_subset, columns=df.columns)

# Print or save the imputed DataFrame as needed
print(imputed_df)


imputed_df.to_csv('imputed_data.csv', index=False)
```

Figure 10: Multiple Imputation in data set

Feature engineering for fraud transaction prediction

```
import math

# Convert 'TransactionDT' to numeric type
All_dataset_transaction['TransactionDT'] = All_dataset_transaction['TransactionDT'].astype('int64')

# Perform floor division and modulo operations
All_dataset_transaction['TransactionHour'] = (All_dataset_transaction['TransactionDT'] // 3600) % 24
All_dataset_transaction['TransactionWeekday'] = ((All_dataset_transaction['TransactionDT'] // (3600 * 24)) + 1) % 7

# Create the TransactionAmountLog feature using the correct column name 'TransactionAmt'
All_dataset_transaction['TransactionAmountLog'] = All_dataset_transaction['TransactionAmt'].apply(lambda x: round(math.log(x + 1e

# Display the head of the modified dataframe to confirm changes
print(All_dataset_transaction[['TransactionHour', 'TransactionWeekday', 'TransactionAmountLog']].head())
```

Figure 11: Feature engineering of TransactionDT

Feature using the P_emaildomain and R_emaildomain columns

```
# Check for missing values in P_emaildomain and R_emaildomain
missing_values = All_dataset_transaction[['P_emaildomain', 'R_emaildomain']].isnull().sum()

# Explore the distribution of email domains in both columns
p_emaildomain_counts = All_dataset_transaction['P_emaildomain'].value_counts()
r_emaildomain_counts = All_dataset_transaction['R_emaildomain'].value_counts()

# Create a new feature that captures whether the purchaser and recipient email domains are the same or different
All_dataset_transaction['EmailMatch'] = All_dataset_transaction['P_emaildomain'] == All_dataset_transaction['R_emaildomain']

# Evaluate if this new feature has any predictive power for fraud
email_match_fraud_rate = All_dataset_transaction.groupby('EmailMatch')['target_variable'].mean()

print('Missing values in P_emaildomain and R_emaildomain:')
print(missing_values)
print('\nDistribution of P_emaildomain:')
print(p_emaildomain_counts.head())
print('\nDistribution of R_emaildomain:')
print(r_emaildomain_counts.head())
print('\nFraud rate by EmailMatch:')
print(email_match_fraud_rate)
```

Figure 12: Feature engineering with email details P and R Domain

```python
# Calculate the correlation matrix to identify highly correlated features
import pandas as pd
import numpy as np
from tqdm.notebook import tqdm

tqdm.pandas()

correlation_matrix = All_dataset_transaction.corr().abs()

# Select upper triangle of correlation matrix
upper = correlation_matrix.where(np.triu(np.ones(correlation_matrix.shape), k=1).astype(np.bool))

# Find index of feature columns with correlation greater than 0.95
highly_correlated_features = [column for column in upper.columns if any(upper[column] > 0.95)]

# Display the features that were dropped
print('Highly correlated features removed:')
print(highly_correlated_features)
```

Figure 13: Multi co-linearity - Features removal

```python
# Training a Lasso Regression model on the train transaction dataset
lasso_model = Lasso(alpha=1.0)
lasso_model.fit(reduced_All_dataset_transaction, target_variable)

# Retrieving the selected features and their corresponding coefficients
selected_features = reduced_All_dataset_transaction.columns[lasso_model.coef_ != 0]
coefficients = lasso_model.coef_[lasso_model.coef_ != 0]

print("Selected features:", selected_features)
print("Coefficients:", coefficients)
```

Figure 14: Feature selection with Lasso Regression

```python
# Calculate the correlation matrix for the features in dataset
corr_matrix = reduced_All_dataset_transaction.corrwith(target_variable)

# Find the absolute value of the correlation coefficients
abs_corr_matrix = np.abs(corr_matrix)

# Select the top 5 features with the highest absolute values of the correlation coefficients
top_features_indices = np.argsort(abs_corr_matrix)[-5:]
top_features = abs_corr_matrix.iloc[top_features_indices]

print("Top 5 features correlating with the target variable (heuristic method):")
print(top_features)
```

Figure 15: Feature selection with Correlation Matrix

```python
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
import pandas as pd
from sklearn.model_selection import train_test_split

# Assuming 'data' contains your dataset with features and 'target_variable'
X = reduced_All_dataset_transaction.drop('target_variable', axis=1)  # Features
y = reduced_All_dataset_transaction['target_variable']  # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply SMOTE only to the training data
smote = SMOTE(sampling_strategy=1.0, random_state=42)  # Equalize classes
X_train_synthetic, y_train_synthetic = smote.fit_resample(X_train, y_train)

# Apply Random Undersampling to the synthetic training data
undersample = RandomUnderSampler(sampling_strategy=1.0, random_state=42)  # Equalize classes
X_train_undersampled, y_train_undersampled = undersample.fit_resample(X_train_synthetic, y_train_synthetic)

# Assuming y_train_undersampled is the balanced target variable after combining SMOTE and Random Undersampling
# Check the class distribution
class_distribution = pd.Series(y_train_undersampled).value_counts(normalize=True)
print(class_distribution)
```

Figure 16: Hybrid - SMOTE and Undersampling to balance the data set

Scaling Object - Feature scaling - Standardization

```python
# Define the scaler object - Feature scaling - Standardization
scaler = StandardScaler()

# Fit the scaler to the training and testing data and transforming it
X_train_scaled = scaler.fit_transform(X_train_undersampled)

print(X_train_scaled)
```

Figure 17: scaling the data set

```python
# Create a Random Forest classifier with 100 trees
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier on the training data
rf.fit(X_train_undersampled, y_train_undersampled)

# Predict the target variable for the testing data
y_pred_rf = rf.predict(X_test)

# Evaluate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred_rf)
precision = precision_score(y_test, y_pred_rf)
recall = recall_score(y_test, y_pred_rf)
f1_score = f1_score(y_test, y_pred_rf)
precision, recall, thresholds = precision_recall_curve(y_test, y_pred_rf)

# Compute the PR-AUC
pr_auc = auc(recall, precision)
```

Figure 18: Model 1 - Batch - Random Forest

```python
# Create a LightGBM classifier
gbm = lgb.LGBMClassifier(num_leaves=31, learning_rate=0.05, n_estimators=100)

# Train the classifier on the training data
gbm.fit(X_train_undersampled, y_train_undersampled)

# Predict the target variable for the testing data
y_pred_gbm = gbm.predict(X_test)

# Evaluate the performance of the model
accuracy_gbm = accuracy_score(y_test, y_pred_gbm)
precision_gbm = precision_score(y_test, y_pred_gbm)
recall_gbm = recall_score(y_test, y_pred_gbm)
f1_gbm = f1_score(y_test, y_pred_gbm)
```

Figure 19: Model 2 - Batch - Gradient Boosting LGBM

```python
# Create an AdaBoost classifier with 100 weak learners
adaboost = AdaBoostClassifier(n_estimators=100, random_state=42)

# Train the classifier on the training data
adaboost.fit(X_train_undersampled, y_train_undersampled)

# Predict the target variable for the testing data
y_pred_em = adaboost.predict(X_test)

#  Print the evaluation metrics
accuracy = accuracy_score(y_test, y_pred_em)
recall = recall_score(y_test, y_pred_em)
precision = precision_score(y_test, y_pred_em)
f1_score = f1_score(y_test, y_pred_em)
precision, recall, thresholds = precision_recall_curve(y_test, y_pred_em)
pr_auc = auc(recall, precision)
conf_matrix = confusion_matrix(y_test, y_pred_em)
```

Figure 20: Model 3 - Batch - Ensemble Adaboost

```python
# Create the Random Forest model
rf_model = RandomForestClassifier()

# Train the model with the undersampled training data
rf_model.fit(X_train_undersampled, y_train_undersampled)

# Placeholder for making predictions in batches with the test data
batch_size = 1000
accuracy_list = []
precision_list = []
recall_list = []
f1_list = []
y_pred_proba_list = []
y_true_list = []
conf_matrix_list = []

# Making predictions in batches with the test data
for i in tqdm(range(0, len(X_test), batch_size)):
    X_batch = X_test[i:i+batch_size]
    y_batch = y_test[i:i+batch_size]
    y_pred = rf_model.predict(X_batch)
    y_pred_proba = rf_model.predict_proba(X_batch)[:,1]
    accuracy_list.append(accuracy_score(y_batch, y_pred))
    precision_list.append(precision_score(y_batch, y_pred))
    recall_list.append(recall_score(y_batch, y_pred))
    f1_list.append(f1_score(y_batch, y_pred))
    y_pred_proba_list.extend(y_pred_proba)
    y_true_list.extend(y_batch)
    conf_matrix_list.append(confusion_matrix(y_batch, y_pred))
    conf_matrix_list.append(conf_matrix)
```

Figure 21: Model 4 - Online - Random Forest - Batch wise data input

```python
# Create the XGBoost model
model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss')

# Train the model with the undersampled training data
model.fit(X_train_undersampled, y_train_undersampled)

# Placeholder for making predictions in batches with the test data
batch_size = 1000
accuracy_list = []
precision_list = []
recall_list = []
f1_list = []
y_pred_proba_list = []
y_true_list = []
conf_matrix_list = []

# Making predictions in batches with the test data
for i in tqdm(range(0, len(X_test), batch_size)):
    X_batch = X_test[i:i+batch_size]
    y_batch = y_test[i:i+batch_size]
    y_pred = model.predict(X_batch)
    y_pred_proba = model.predict_proba(X_batch)[:,1]
    accuracy_list.append(accuracy_score(y_batch, y_pred))
    precision_list.append(precision_score(y_batch, y_pred))
    recall_list.append(recall_score(y_batch, y_pred))
    f1_list.append(f1_score(y_batch, y_pred))
    y_pred_proba_list.extend(y_pred_proba)
    y_true_list.extend(y_batch)
    conf_matrix = confusion_matrix(y_batch, y_pred)
    conf_matrix_list.append(conf_matrix)
```

Figure 22: Model 5 - Online - XGBoost - Batch wise data input

```python
print(f"The accuracy of the AdaBoost classifier is {accuracy}.")
print(f"The recall of the AdaBoost classifier is {recall}.")
print(f"The precision of the AdaBoost classifier is {precision}.")
print(f"The F1 score of the AdaBoost classifier is {f1_score}.")
print(f"The PR-AUC of the AdaBoost classifier is {pr_auc}.")
print("Confusion Matrix:")
print(conf_matrix)
```

Figure 23: Evaluation Metrics

```python
# Plot Precision-Recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall_gbm, precision_gbm, marker='.')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve for LightGBM')
plt.grid(True)
plt.show()
```

Figure 24: Plotting the PR Curve

Model	Accuracy	Precision	Recall	F1 Score	PR-AUC
Random Forest	97.41	72.67	43.47	51.68	59.59
AdaBoost	86.86	13.37	48.54	20.97	31.88
LightGBM	92.74	23.24	44.32	30.49	34.78
Online Random Forest	97.42	73.50	43.01	53.89	56.83
Online XGBoost	95.95	42.92	42.21	42.24	42.05

Table 2: Model Performance Metrics

Figure 25: Results