Configuration Manual

MSc Research Project

MSc in AI for Business

Thi Thuy Hang Nguyen

Student ID: 23347325

School of Computing

National College of Ireland

Supervisor: Rejwanul Haque

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Thi Thuy Hang Nguyen

Student ID: 23347325

Programme: MSCAIBUS1 **Year:** 2024-2025

Module: Practicum

Lecturer: Rejwanul Haque

Submission Due Date: 11th August 2025

Project Title: Cost – Sensitive Hyperparameter Tuning for XGBoost in Credit Card Fraud Detection

Word Count: **1187** **Page Count:** 7

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: Thi Thuy Hang Nguyen

Date: 12/09/2025

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:	Thi Thuy Hang Nguyen
Date:	12/09/2025
Penalty Applied (if applicable):	

Configuration Manual

Thi Thuy Hang Nguyen
Student ID: 23347325

1 Introduction

This configuration manual provides guidance for this project. Its also outlines the process for designing and implementing the project.

2 Hardware configuration

- **RAM:** 16.0 GB
- **OS:** Windows 11 Pro
- **Processor:** 11th Gen Intel(R) Core (TM) i5-1135G7
- **Technology required:** Python, VS code

3 Setup for Configuration

3.1 Software Configuration

Visual Studio Code

Visual Studio code (VS code) is a free, lightweight, and cross – platform code editor developed by Microsoft. It's widely used by developers for coding, debugging, and project management.

Key features:

- **IntelliSense:** Smart code completion and sytax highlighting.
- **Built – in Debugger:** Set breakpoints, inspect variables, and debug code easily.
- **Extensions:** Thousands of plugins for languages, tools, and frameworks.

- **Integrated Terminal:** Run commands without leaving the editor.
- **Git Integration:** Manage source control directly within VS code.
- **Customization:** Themes, keybindings, and layout is fully customizable.
- **Live Sare:** Real – time collaboration with others.

Main Functions in Development:

- Writing and editing code
- Running and debugging applications
- Managing files and folders in a project
- Using Git for version control
- Installing tools and extensions to support various programming needs.

3.2 Libraries Configuration

The project employs a variety of Python libraries to facilitate data processing, visualization, machine learning, and model assessment:

(1) Pandas and numpy: Utilized for data manipulation, array handling, and numerical calculations.

(2) Scikit – learn (sklearn): Offers utilities for dividing datasets (train_test_split), feature scaling (StandardScaler), generating artificial datasets (make_classification, load_iris), training models (RandomizedSearchCV), and assessing performance through metrics such as classification_report, confusion_matrix, roc_auc_score, and display functions including ConfusionMatrixDisplay and RocCurveDisplay.

(3) Matplotlib.pyplot and seaborn: Employed for data visualization, such as distribution plots, correlation plots, and evaluation results.

(4) XGBoost (XGBClassifier): An efficient gradient boosting framework utilized for developing and optimizing the classification model.

(5) Imbalanced – learn (SMOTE): Utilized for balancing class imbalance in the dataset through oversampling of the minority class based on the SMOTE method.

4 Implementation

(1) Importing all libararies:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, auc, Confusion
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import make_classification
from xgboost import XGBClassifier
from imblearn.over_sampling import SMOTE
```

[1] ✓ 6.7s Python

Figure 1. Importing all library

```
pip install xgboost
```

[2] ✓ 6.0s Python

Figure 2. Install XGBoost

```
pip install xgboost scikit-learn matplotlib
```

[3] ✓ 6.0s Python

Figure 3. Install XGBoost scikit – learn

```
from xgboost import XGBClassifier
```

[4] ✓ 0.0s Python

Figure 4. Importing the XGBClassifier from the XGBoost Library

(2) Loading data

```
# Load data
df_data = pd.read_csv('../creditcard.csv')
```

[5] ✓ 1.3s Python

Figure 5. Loading data

(3) Preprocessing data

```
print(df_data.info())
df_data.head()
```

[11] Python

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
```

Figure 6. Preprocessing data

The data utilized for this research is credit card transactions performed by cardholders in Europe in September 2013. It contains data for a two-day interval, 284,807 transactions, with only 492 being fraudulent. This constitutes an extremely imbalanced distribution, with fraudulent cases making up only 0.172% of all cases. In order to maintain user privacy, the data has been anonymized with Principal Component Analysis (PCA), so the original feature names and values are not interpretable. Nevertheless, the features "Amount" and "Time" are kept in their original state and are standardized through a normalization method to be on the same scale as the PCA-transferred features. The target feature is binary, where "0" is a normal transaction and "1" is fraud.

(4) Baseline model

```
# Scale Amount and Time
scaler = StandardScaler()
df_data['Amount'] = scaler.fit_transform(df_data[['Amount']])
df_data['Time'] = scaler.fit_transform(df_data[['Time']])
```

[8] Python

```
# Split features and target
X = df_data.drop('Class', axis=1)
y = df_data['Class']

# Train/test split with stratification to keep class balance
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=
```

[9] Python

Figure 6. Feature Scaling and Stratified Train-Test Split in Baseline Model

During this stage, the Amount and Time features are scaled using the StandardScaler from scikit-learn, thus ensuring that they are on a similar scale as the PCA-transformed features. Since the original dataset includes features with a wide range of magnitudes, normalization helps improve the model's convergence and stability. After scaling, the dataset is split into independent variables (X) and dependent variable labels (Y), with the target column Class indicating if a transaction is fraudulent (1) or not (0). The dataset is then divided into training and testing sets using an 80/20 split while preserving the original class distribution using stratification. This ensures that the model is trained using a representative subset of both fraudulent and legitimate instances, which is important for effectively dealing with imbalanced data situations.

(5) Output baseline model

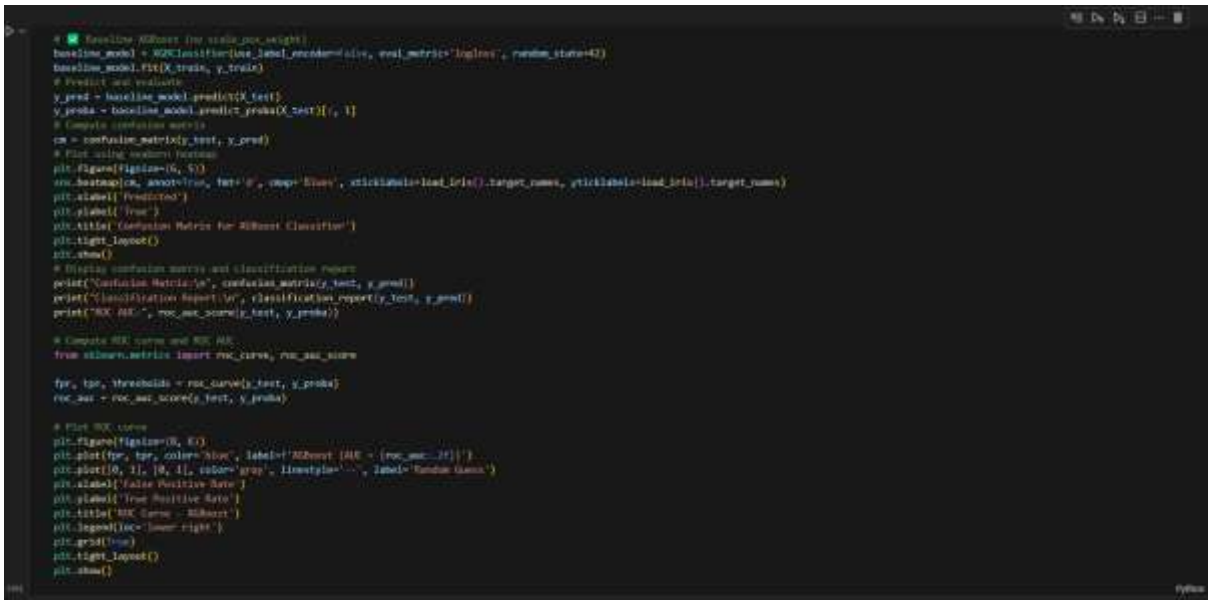

Figure 7. Baseline model (XGBoost without cost tunning)

For the baseline experiment, we trained an XGBoost classifier without any cost-sensitive adaptation—i.e., we left the scale_pos_weight parameter at its default value. This approach gives us a baseline comparison for further comparison. We instantiated the model with default parameters, setting logloss as the evaluation metric and disabling label encoding for compatibility with binary classification. After training the model on the training dataset, we generated predictions on the test dataset and saved both the predicted class labels and class probabilities for downstream evaluation.

In order to assess the model performance, a confusion matrix was computed and visualized as a heatmap. The matrix provides a notion of the number of correctly and wrongly predicted transactions for both fraudulent and legitimate classes. Furthermore, a classification

report was generated that shows key metrics such as precision, recall, and F1-score for both classes. The ROC AUC score—yet another key metric in imbalanced classification—was also calculated to measure the ability of the model in discriminating between the two classes at various threshold levels.

Lastly, the Receiver Operating Characteristic (ROC) curve was drawn to show the trade-off between the true positive rate (recall) and the false positive rate. The area under the ROC curve (AUC) gives a threshold-independent measure of classification performance. This baseline model serves as a reference point against which later models with cost-sensitive adjustments and resampling techniques can be compared.

(6) SMOTE and cost sensitive learning to improve result

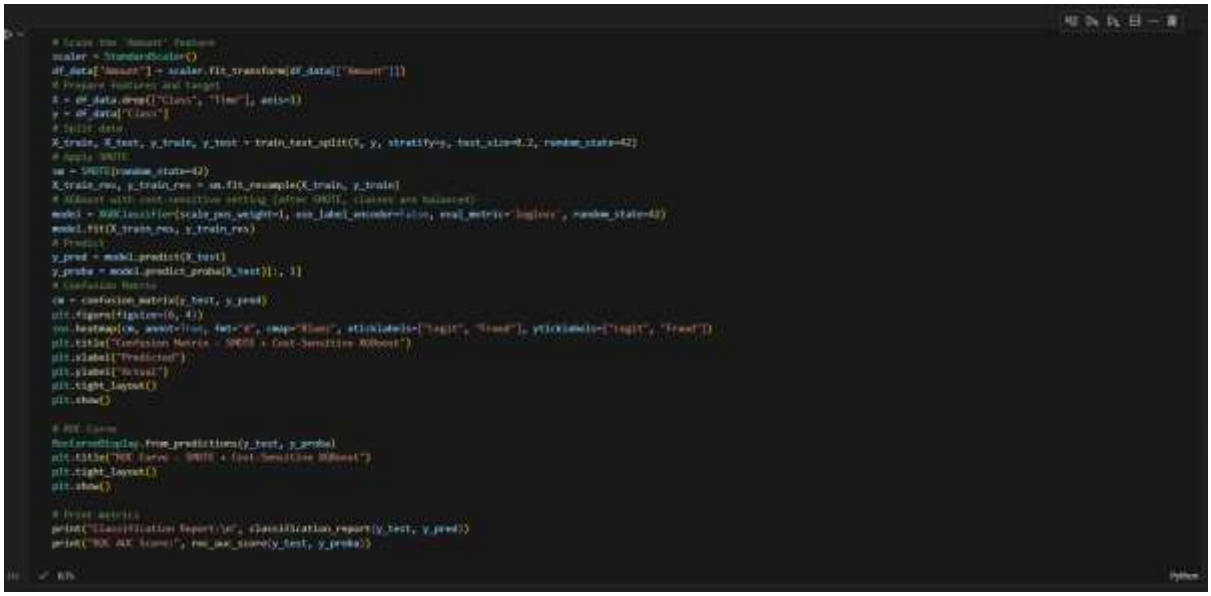

Figure 8. SMOTE and cost sensitive XGBoost

In this case, the XGBoost model was trained using the cost-sensitive learning approach combined with the SMOTE oversampling technique to combat the extreme class imbalance inherent in credit card fraud data. The Amount feature was standardized using StandardScaler to be similar in scale to the PCA-transformed features. The Time feature was dropped, as it does not contribute directly to the classification in this case.

The data was divided into training and test sets by stratified sampling to maintain the class distribution of the original data in both sets. To counter class imbalance in the training data, the Synthetic Minority Over-sampling Technique (SMOTE) was employed. SMOTE creates synthetic samples of the minority class (fraudulent cases) by interpolating between examples, creating a more balanced training distribution without modifying the test set.

After resampling, the XGBoost model was trained with a scale_pos_weight parameter of 1 because the classes are already balanced after SMOTE. This pairing allows the model to maintain sensitivity to the minority class while still being able to take advantage of the strict learning procedure of gradient boosting. The model's predictions on the test set were evaluated on a number of metrics.

A confusion matrix was created to visualize the counts of correct and incorrect classifications for both the fraud and legitimate classes. A classification report was printed out to provide an overview of important performance metrics, such as precision, recall, and F1-score. Lastly, the ROC curve was plotted to examine the model's class discriminatory power across thresholds, and the ROC AUC score was calculated as a threshold-independent performance metric.

This combined method of synthetic oversampling and cost-sensitive learning aims to enhance the recall of the model on fraudulent transactions without allowing false positives to get out of hand, a more realistic solution for fraud detection in actual financial systems.