

ABSTRACT

After Covid-19 pandemic, the popularity of e-commerce has increased exponentially which made possible to carryout transaction anywhere by anyone during lockdown through online payment system. However, increasing online transaction has also increased the risk for online frauds. Frauds like credit card frauds are easy which aggregate to enormous losses annually. Therefore, it is crucial to implement the mechanisms that detect such malicious activities. Hence, many machines learning models have been developed by researchers to solve this problem. In this project, two different models – Logistic Regression (LR) and Extreme Gradient Boosting Model on Decision Tree (DT) are used and comparative analysis based on accuracy and detection rate of these models has been carried out. Genetic Algorithm is used to optimized the hyperparameters of Gradient Boosting Model. The conclusion of our study focuses on the comparison of different models and their performance for credit card fraud detection.

Keywords: Logistic Regression, Extreme Gradient Boosting, Decision Tree, Genetic Algorithm, Hyperparameters

Table of contents

ACKNOWLEDGEMENT	i
ABSTRACT.....	ii
Table of Figures	vi
List of Abbreviation.....	ii
CHAPTER 1	1
1. Introduction.....	1
1.1 Introduction to credit card fraud detection.....	1
1.2 Statement of problem	1
1.3 Objectives.....	2
1.4 Scope	2
1.5 Limitation	2
CHAPTER 2	3
2. Literature Review.....	3
CHAPTER 3	5
3. Methodology	5
3.1 Data Collection.....	5
3.2 System Requirement	5
3.3 Data Preprocessing.....	5
3.4 Architecture of model.....	6
3.4.1 Logistic Regression	6
3.4.2 Gradient Boosting Model	7
3.4.3 Decision Tree.....	10
3.4.4 Genetic Algorithm for tuning hyperparameters.....	11
3.5 Workflow	12

3.6	Result Evaluation Metrics	13
3.6.1	Accuracy	13
3.6.2	Precision	13
3.6.3	Recall	13
3.6.4	F1 Score	13
3.6.5	Confusion matrix	13
3.6.6	AUC-ROC curve	14
CHAPTER 4	15
4.	Implementation	15
4.1	Programming language and Libraries	15
4.2	Splitting the datasets.....	15
4.3	Algorithm.....	15
4.3.1	Logistic Regression.....	15
4.3.2	Boosting Model (XGBoost).....	16
4.3.3	Genetic Algorithm	18
CHAPTER 5	19
5.	Result analysis and Discussions.....	19
5.1	Performance Analysis of Logistic Regression	19
	Unbalanced Dataset	19
	Balanced Dataset (using Library)	20
	Balanced Datasets (from scratch without using Library)	21
5.2	Performance Analysis of Boosting Model	22
5.3	Comparison of different models on different datasets	23
CHAPTER 6	26
6.	Conclusion and Future recommendation	26

6.1 Conclusion	26
6.2 Future Works	26
References	27
Appendices.....	28

Table of Figures

FIGURE 1: LOGISTIC REGRESSION	7
FIGURE 2: GRADIENT BOOSTING MODEL	9
FIGURE 3: DECISION TREE	10
FIGURE 4 : GENETIC ALGORITHM.....	11
FIGURE 5 : WORKFLOW DIAGRAM	12
FIGURE 6: CONFUSION MATRIX	13
FIGURE 7 : UNBALANCED DATASET AUC-ROC CURVE	20
FIGURE 8: BALANCED AUC-ROC CURVE.....	21
FIGURE 9: BOOSTING MODEL AUC- ROC CURVE	22
FIGURE 10: BAR CHART OF CREDITCARD1.CSV	23
FIGURE 11: RESULT OF CREDITCARD1.CSV	23
FIGURE 12: BAR CHART OF CREDITCARD2.CSV	24
FIGURE 13: RESULT OF CREDICARD2.CSV	24
FIGURE 14: DATASETS PART1	28
FIGURE 15: DATASETS PART2	28
FIGURE 16: IMBALANCE DATA	28
FIGURE 17: BALANCED DATA.....	29
FIGURE 18: LOGISTIC REGRESSION ALGORITHM CODE	30
FIGURE 19: GENETIC ALGORITHM INITIALIZATION CODE	31
FIGURE 20: GENETIC ALGORITHM FITNESS FUNCTION CODE.....	32
FIGURE 21: GENETIC ALGORITHM TRAIN POPULATION CODE	32
FIGURE 22: GENETIC ALGORITHM PARENT SELECTION CODE	33
FIGURE 23: GENETIC ALGORITHM CROSSOVER UNIFORM CODE.....	33
FIGURE 24: GENETIC ALGORITHM MUTATION CODE	34
FIGURE 25: MAIN PAGE CODE.....	35

List of Abbreviation

Abbreviation	Definition
AI	Artificial Intelligence
AUC	Area Under ROC Curve
CPU	Central Processing Unit
DT	Decision Tree
FN	False Negative
FP	False Positive
GBM	Gradient Boosting Machine
GPU	Graphics Processing Unit
GA	Genetic Algorithm
LR	Logistic Regression
ML	Machine Learning
PCA	Principle Component Analysis
ROC	Receiver Operating Characteristic Curve
SMOTE	Synthetic Minority Oversampling Technique
TP	True Positive
TN	True Negative
XGBoost	Extreme Gradient Boosting

CHAPTER 1

1. Introduction

1.1 Introduction to credit card fraud detection

Credit cards are widely used for purchasing goods and services, and cash withdrawal. Credit card fraud is a type of identity theft that involves the unauthorized use of credit card information to charge purchases to the account or withdraw funds from it. With the increasing number of credit card transactions, cases of credit card fraud are also on the rise. In Nepal, there have been several cases of credit card fraud, highlighting the vulnerability of credit card transactions in the country.

To address these threats and risks, machine learning algorithms have been proposed as a means of detecting and preventing fraudulent activities. The proposed system aims to detect credit card fraud and prevent unauthorized credit card transactions. Specifically, the performance of various credit card fraud detection techniques is compared to identify the most effective approach for detecting fraudulent transactions. By conducting this analysis, it is expected that valuable insights will be gained into the effectiveness of different fraud detection methods, helping to inform decisions about which approach is most suitable for detecting fraudulent transactions.

Overall, this project aims to provide a better understanding of the challenges and opportunities associated with credit card fraud detection in Nepal. It is expected that the results of this study will contribute to the development of more effective and efficient methods for detecting fraudulent transactions, ultimately leading to improved security and reduced financial losses for consumers and businesses alike.

1.2 Statement of problem

The issue of credit card fraud is becoming a big concern for banks and customers. As people use credit cards more often, the chance of fraudulent transactions also increases. The old ways of finding fraud, such as rule-based systems, can be easily bypassed by skilled fraudsters, making it necessary to come up with new ways of finding fraudulent transactions. The purpose

of this project is to study how well different techniques work for finding credit card fraud, particularly logistic regression and gradient boosting machines (GBM). This project analyzes about how these techniques perform on a dataset of credit card transactions and check their ability to correctly identify fraudulent transactions. The results of this analysis will provide insight into the strengths and limitations of these techniques and inform decisions about which approach is most suitable for detecting credit card fraud.

1.3 Objectives

The main objectives of this project work are:

- To compare the performance of Logistic Regression and Optimized GBM algorithm in detecting credit card fraud based on its performance metrics such as accuracy, precision, recall, F1 score, and AUC-ROC curve.
- To determine the best approach for credit card fraud detection.

1.4 Scope

The Credit Card Fraud Detection project using Logistic Regression, XGBoost, and Genetic Algorithm aims to develop a system for detecting fraudulent credit card transactions using Python programming language and various libraries. The performance of each algorithm is being evaluated using accuracy, precision, recall, and F1 score, and the results are being compared to identify the best-performing algorithm.

1.5 Limitation

Some limitations are:

- The project does not cover the implementation of the credit card fraud detection system in a real-world production environment and may not be able to detect all cases of credit card fraud due to the constantly evolving nature of fraud tactics.
- The study is limited only on 3 algorithms which may be affected by the quality of the dataset used for training and testing the models.

CHAPTER 2

2. Literature Review

Hanzeng Wang in April 2022 applied three machine learning models, decision tree, random forest, and AdaBoost, to a dataset of 2 million simulated transactions from Kaggle. Results were evaluated using classification reports, confusion matrix, ROC curve, and AUC score. The study found that the random forest model performed best, with a macro average F1 score of 0.78 and an accuracy of 75% in predicting fraud. However, limitations such as the use of a simulated dataset and a limited number of models were acknowledged and future research was suggested to focus on larger datasets and other models for improved accuracy. [1]

Faroque Ahmed and Rittika Shamsuddin (2021), the effectiveness of six machine learning techniques (Logistic Regression, Support Vector Machine, Naive Bayes, Random Forest, Decision Tree, and K-nearest neighbor) was evaluated to detect credit card fraud using five types of datasets (imbalanced, under-sampled, over-sampled, SMOTE, and ADASYN). The best combination of techniques was selected based on five performance evaluation criteria (accuracy, AUC, precision, recall, and F1-score). The study found that the Random Forest classifier with over-sampling technique performed the best, showing 99.99% accuracy, precision, AUC, F1-score, and 100% recall rate. It was suggested that this method could be used by financial institutions to minimize credit card frauds. [2]

K R Sumana et al. analyzed the use of Machine Learning and Deep Learning algorithms for credit card fraud detection. The study compared traditional ML algorithms (Logistic Regression, SVM, Naive Bayes, Random Forest, and Decision Trees) and found that they performed better than an Artificial Neural Network (ANN) with an accuracy of 65.67%. Results were evaluated using precision, accuracy and recall. [3]

Eman H. A studied about the effective economic planning and decision-making in the oil and gas industry. Numerous techniques have been used for this purpose, but artificial intelligence (AI)-based models have shown promising results in different sectors and stages of oil exploration and production. The research suggests an enhanced gradient boosting model,

named GA-GB, which applies a genetic algorithm (GA) for predicting crude oil production. The model was effectively utilized to predict crude oil production, oil prices, and demand in different countries, encompassing both major producers and those with lower production rates. The proposed GA-GB model outperformed five traditional regression models, including Bagging, KNN, MLP, RF, and Lasso regressors, in terms of different performance metrics such as MAE, MSE, MedAE, RMSE, and R². This model can improve the planning of concerned departments by forecasting oil production, prices, and demand. However, data availability remains a limitation, and future research should focus on including essential factors and analyzing their impact on oil production. [4]

XinYing Chew et al. investigated the use of seven hybrid machine learning models to detect credit card fraud using a real-world dataset. The findings of the study indicated that the Adaboost + LGBM hybrid model had superior performance in detecting credit card fraud compared to the other hybrid models tested. The study suggests that the use of hybrid models can yield major advantages over state-of-the-art models in detecting fraudulent activities in the financial sector. Future work could focus on exploring different types of hybridization and algorithms in the credit card domain and extending the use of hybrid models to other datasets. [5]

CHAPTER 3

3. Methodology

3.1 Data Collection

The data for this study was collected from the online community Kaggle. It consists of a dataset of credit card transactions made by European cardholders in September 2013, including 284,807 transactions with 492 identified as fraudulent. The dataset is imbalanced, with a fraudulent rate of 0.172%. The datasets are divided into two datasets creditcard1.csv creditcard2.csv, where creditcard.csv contains 262 fraud data and 142141 non-fraud datasets whereas creditcard2.csv contains 230 fraudulent data and 142174 non-fraudulent data. The dataset consists of numerical input variables transformed using PCA, with the features 'Time' and 'Amount' not transformed. The response variable 'Class' is binary, with a value of 1 for fraudulent transactions and 0 for non-fraudulent transactions. Confidentiality issues prevent the disclosure of original features or additional background information about the data.

3.2 System Requirement

Hardware Requirement

Processor	:	Pentium Dual Core 2.3GHz
Hard Disk	:	250 GB or Higher
RAM	:	4 GB or Higher

Software Requirements

Operating System	:	Window 7 or Higher
Languages used	:	Python 3.6 or Above
Tools	:	Visual Studio Code, Jupyter Notebook

3.3 Data Preprocessing

Although the datasets don't contain any missing data or value but the dataset is highly imbalanced. The datasets require the balancing of data which can be performed by using different techniques such as oversampling the minority class or under sampling the majority class.

3.4 Architecture of model

Architecture of model provides the structure and the components used to build that machine learning models. The architecture of our machine learning model depends on the type of model and the problem it is being used to solve.

3.4.1 Logistic Regression

In credit card fraud detection, the goal of the logistic regression model would be to predict whether the given credit card transaction is fraudulent or not, based on a set of input features. It is divided into 5 different parts.

1. **Input features:** The logistic regression model takes a set of input features as input. These features might include characteristics of the transaction (such as the amount, type, location, and time of day), as well as features derived from the credit card holder (such as their spending patterns and demographic information).
2. **Linear combination:** The input features are combined linearly using a set of coefficients (also known as weights). The linear combination is computed as follows:

$$\text{Linear combination} = w_1x_1 + w_2x_2 + \dots + w_nx_n$$

Where x_1, x_2, \dots, x_n are the input features, and w_1, w_2, \dots, w_n are the corresponding coefficients.

3. **Sigmoid function:** The output of the linear combination is passed through a sigmoid function, which maps the output to a value between 0 and 1. The sigmoid function is defined as follows:

$$\text{Sigmoid}(x) = 1 / (1 + e^{(-x)})$$

4. **Probability prediction:** The predicted probability that the input belongs to the positive class (i.e., fraudulent transactions) is interpreted from the output of the sigmoid function.
5. **Decision boundary:** A decision boundary is chosen, typically at a probability of 0.5. Transactions with a predicted probability greater than or equal to the decision boundary are classified as fraudulent, while those with a predicted probability below the decision boundary are classified as non-fraudulent.

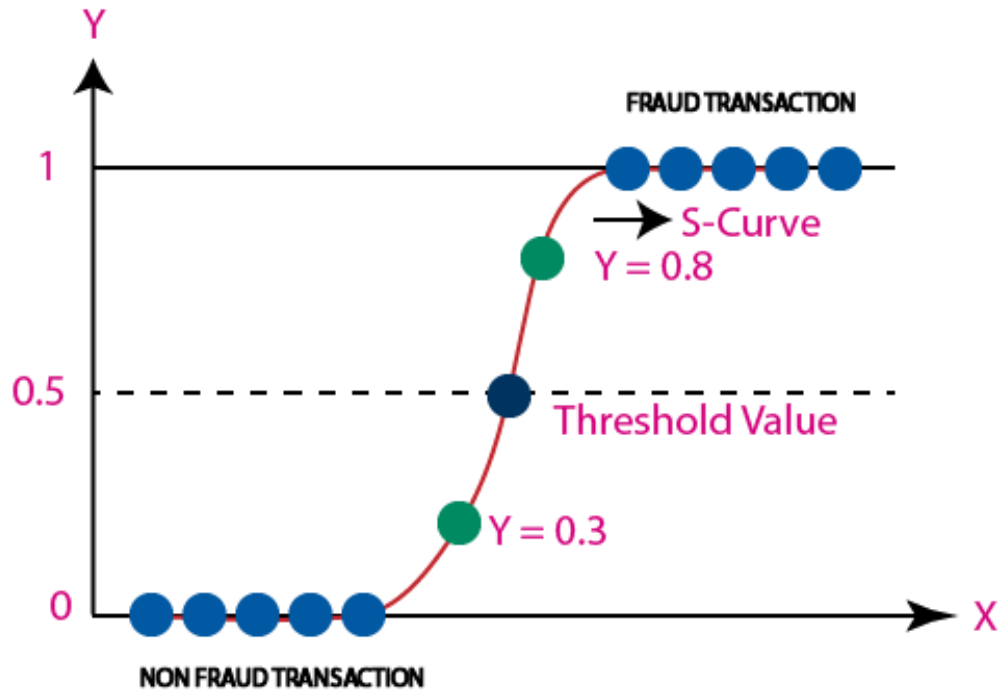


Figure 1: Logistic Regression

Types of Logistic Regression:

- **Binomial Logistic Regression:** In binomial logistic regression, the dependent variable can only have two possible values, typically represented as 0 and 1, true/false, pass/fail or yes/no. The aim of this regression is to predict the probability of an event occurring based on one or more independent variables. In our project, binomial logistic regression is being used as it is to classify whether the transaction is fraud or not.
- **Multinomial Logistic Regression:** Multinomial logistic regression is used when the dependent variable has three or more unordered categories such as: C, C++ or C#. And the goal is to predict the probability of an observation belonging to each category based on independent variables.
- **Ordinal Logistic Regression:** It is similar to multinomial but here the ordered types of dependent variables are used such as: first, second or third.

3.4.2 Gradient Boosting Model

The boosting algorithms are used for building predictive models for regression and classification tasks. It is also known as Ensemble method which combines several weak learners into a strong learner. In our case, the weak learners are a decision tree as the basic

building blocks of GBM is decision tree. Gradient Boosting functions by adding predictors to an ensemble sequentially, with each one correcting the errors of its predecessor. Instead of adjusting instance weights at each iteration, it aims to fit the new predictor to the residual errors produced by the prior predictor.

It consists of 3 elements:

1. **Loss Function:** Loss Function measures the difference between the predicted value and the actual value for a given inputs. There are different types of loss function which are used according to the given problem. In the case of credit card fraud detection, logistic loss function is commonly used.

The logistic loss function is defined as:

$-\log(p(y=1|x))$ if $y = 1$

$-\log(1 - p(y=1|x))$ if $y = 0$

Where $p(y=1|x)$ is the predicted probability of the positive class i.e. fraud given the input x , and y is the true class label i.e. 1 for fraud and 0 for non-fraud.

2. **Weak Learners:** Decision trees are used as a weak learners in gradient boosting model. A decision tree is a tree-like model that is composed of internal nodes, which represent the input features, and leaf nodes which represent the predicted class labels. At each internal nodes, decision tree splits the data based on the value of the input feature and creates a new child node for each possible value. The process is repeated until a leaf node is reached, which represents a predicted class label. A special case of decision tree with only one splits is known as a tree stump. In simple application, tree stump provides considerably accurate results.

The decision tree algorithm works by recursively partitioning the feature space into smaller and smaller regions, each associated with a predicted class label which is used to reduce the impurity of datasets.

3. **Additive Model:** It is built by iteratively adding decision trees to the ensemble. At each iteration, a new decision tree is trained to correct the mistakes of the previous trees in the ensemble.

The final prediction of an additive model can be represented mathematically as:

$$F(x) = f_0(x) + f_1(x) + \dots + f_n(x)$$

Where $F(x)$ is the final prediction, $f_0(x)$, $f_1(x)$, ..., $f_n(x)$ are the predictions of the individual decision trees in the ensemble and x is the input.

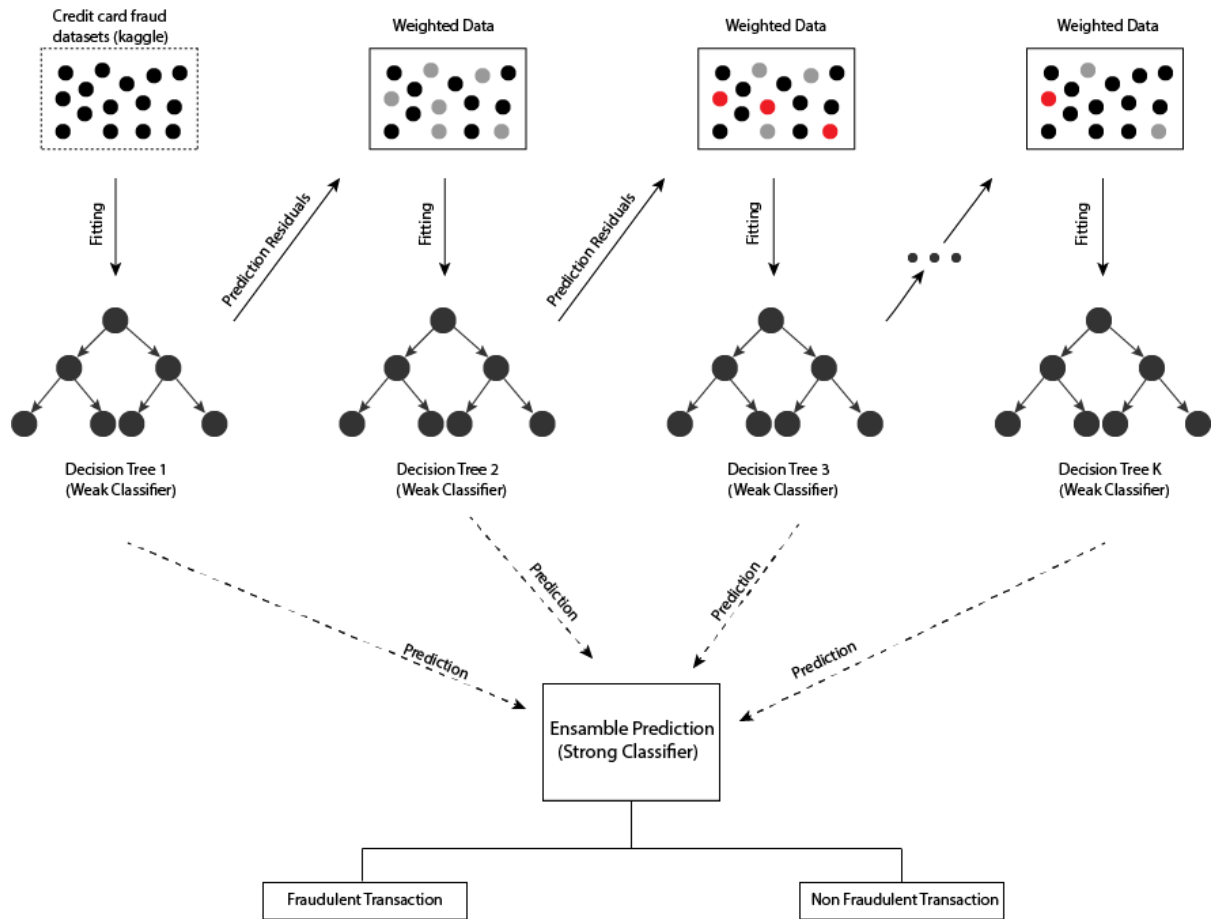


Figure 2: Gradient Boosting Model

There are different forms of boosting model , such as:

- XGBoost
- Gradient Boosting
- LightGBM

XGBoost is chosen for our datasets as it is the enhanced version of gradient boosting and is designed to be highly efficient, making it fast and scalable. It can handle very large datasets with millions of examples and thousands of features, and can be parallelized across multiple

CPUs or GPUs to speed up training. It includes several regularization techniques to prevent overfitting and improve generalization performance.

3.4.3 Decision Tree

The decision tree algorithm is a popular machine learning algorithm that is used for both classification and regression tasks. It works by recursively partitioning the dataset into smaller and smaller subsets based on the values of different input features until the resulting subsets are as pure as possible in terms of the target variable.

The basic idea behind decision trees is to create a tree-like model of decisions and their possible consequences. The tree is constructed by recursively splitting the dataset based on the value of a particular feature at each node. The splitting criteria are chosen to maximize the information gain, which measures the reduction in entropy (or increase in purity) achieved by the split.

Each internal node of the tree represents a decision based on a specific feature, and each leaf node represents a prediction or outcome. To make a prediction for a new data point, we simply start at the root node and follow the path down the tree based on the values of the input features until we reach a leaf node, which gives us the predicted value.

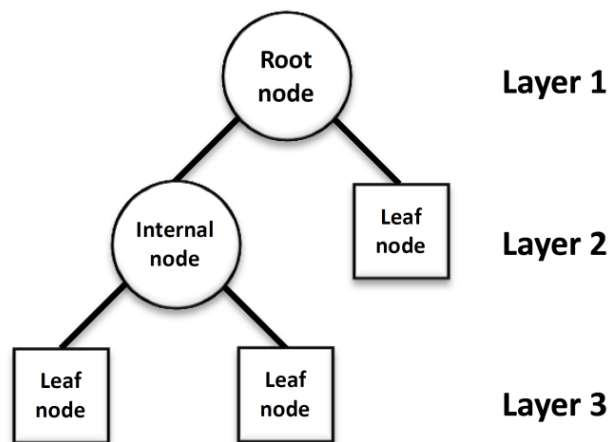


Figure 3: Decision Tree

3.4.4 Genetic Algorithm for tuning hyperparameters

Genetic Algorithm are adaptive heuristic search optimization algorithm inspired from the idea of natural selection and genetics which occurs in biology. It simulates “survival of fittest” theory of Charles Darwin where the best fit solutions are selected for the further breeding. It is used to solve the optimization problems in machine learning.

Gradient Boosting Model have several hyperparameters. Using all those hyperparameters can effect on its performance which can leads to underfitting or overfitting problems. In case of not tuning the hyperparameters, some of the frameworks and libraries can provide the default hyperparameters which might not be suitable for the given specific datasets or problem.

Advantages of tuning hyperparameters:

- It enhances the performance of model by reducing the overfitting and underfitting problem.
- It captures the complex relationship between features and the target variables.
- It reduces helps the training time.
- It improves the interpretability of the model.

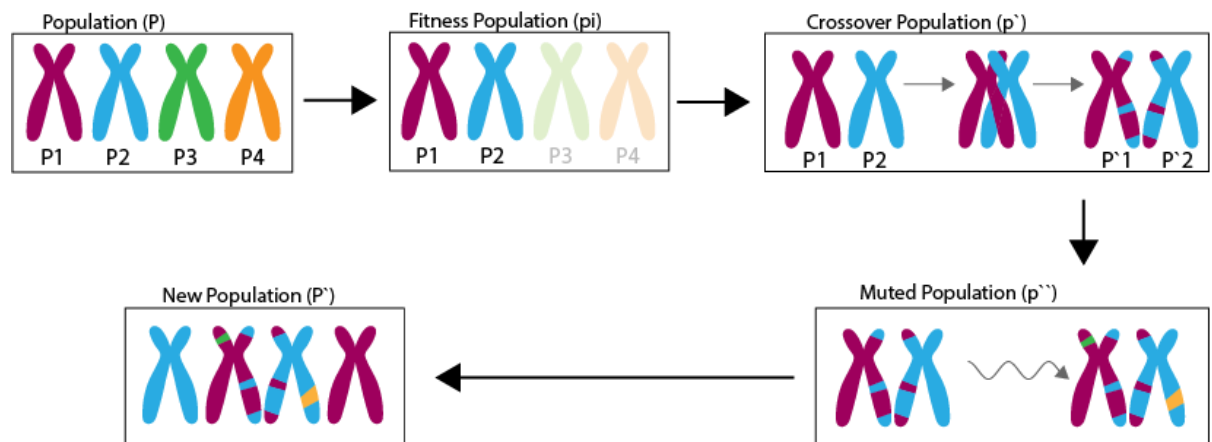


Figure 4 : Genetic Algorithm

The different elements of Genetic Algorithm are:

1. **Chromosomes:** Genetic algorithms use a binary string called a chromosome to represent the parameters of an optimization problem. The optimization process involves improving the population of chromosomes over multiple generations through genetic operations.

2. **Population:** The population in genetic algorithms comprises chromosomes representing potential solutions to the problem. It evolves over generations through selection, crossover, and mutation to improve the population's fitness.
3. **Fitness function:** The fitness function evaluates how well each chromosome solves the optimization problem and selects higher-performing chromosomes for reproduction in the next generation.
4. **Selection:** Selection involves picking the fittest chromosomes from the current population to be parents in the next generation.
5. **Crossover:** During crossover, genetic information is exchanged between two parent chromosomes to create offspring chromosomes, inspired by the natural genetic recombination process that happens during sexual reproduction.
6. **Mutation:** Mutation randomly alters a small portion of a chromosome to introduce new genetic information into the population, promoting diversity and preventing the population from becoming trapped in local optima.

3.5 Workflow

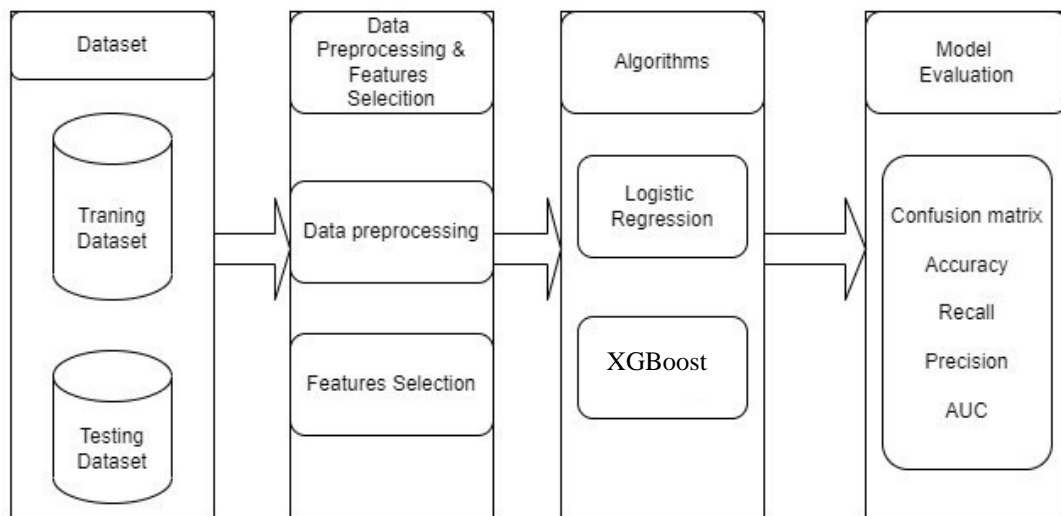


Figure 5 : Workflow Diagram

3.6 Result Evaluation Metrics

Some varieties of ways that can be used to evaluate results are listed below:

3.6.1 Accuracy

The proportion of correctly classified instances over the total number of instances.

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN)$$

3.6.2 Precision

The ratio of true positive predictions to the total number of positive predictions made.

$$\text{Precision} = TP / (TP + FP)$$

3.6.3 Recall

The proportion of true positives that were correctly identified, while ignoring false negatives.

$$\text{Recall} = TP / (TP + FN)$$

3.6.4 F1 Score

It is the harmonic mean of precision and recall, giving equal weight to both metrics and providing a balanced evaluation of the model's effectiveness.

$$\text{F1 score} = 2 * (\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})$$

3.6.5 Confusion matrix

The performance of a classifier can be summarized using a table that compares the predicted labels with the actual labels. This table presents the counts of true positives, false positives, true negatives, and false negatives.

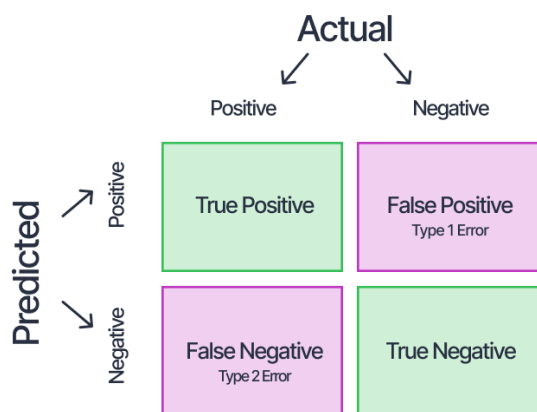


Figure 6: Confusion Matrix

Where,

True Positive (TP): The model correctly predicts a positive class (e.g., fraud) when the true label is positive (i.e., the transaction is actually fraudulent).

False Positive (FP): The model incorrectly predicts a positive class (e.g., fraud) when the true label is negative (i.e., the transaction is actually legitimate).

True Negative (TN): The model correctly predicts a negative class (e.g., legitimate) when the true label is negative (i.e., the transaction is actually legitimate).

False Negative (FN): The model incorrectly predicts a negative class (e.g., legitimate) when the true label is positive (i.e., the transaction is actually fraudulent).

3.6.6 AUC-ROC curve

The AUC-ROC (Area Under the Receiver Operating Characteristic Curve) is a commonly used performance metric in binary classification tasks like fraud detection. This metric measures a model's ability to differentiate between positive and negative classes by plotting the true positive rate against the false positive rate. The AUC-ROC score ranges from 0 to 1, where a score of 1 represents perfect discrimination between the two classes. In the context of fraud detection, a higher AUC-ROC score indicates a better ability to detect fraud, correctly identifying more fraudulent transactions as positive and more non-fraudulent transactions as negative.

The AUC-ROC score is a valuable tool for evaluating the effectiveness of fraud detection models, enabling developers to optimize and refine their models to improve their accuracy and performance. By comparing the AUC-ROC scores of different models, developers can select the best model for a given dataset and application. Overall, the AUC-ROC score is an essential metric for measuring the success of fraud detection models, allowing developers to make informed decisions when designing and deploying these models.

CHAPTER 4

4. Implementation

4.1 Programming language and Libraries

The implementation of all the algorithms and programs was carried out using the Python programming language, version 3.10.10. The software development environment used for this task comprised Visual Studio Code and Jupyter Notebook. NumPy, pandas, seaborn and matplotlib libraries of python has been used to manipulate dataset and make it easier for analysis and visualization. For splitting the data into training and testing sets and evaluating accuracy scores, Scikit-learn library has been utilized. Additionally, the Imbalance-learn library, which relies on Scikit-learn has also been used to address balances the datasets. In order to create the user interface (UI), the Tkinter Python library has been utilized.

4.2 Splitting the datasets

Here are the steps used to split the credit card dataset into two parts while maintaining the ratio of fraud and non-fraud transactions of original datasets:

1. Read in the credit card dataset using pandas.
2. Split the dataset into fraud and non-fraud data frames.
3. Shuffle the indices of the fraud and non-fraud datasets using NumPy.
4. Split each dataset into two parts and create two subsets for each dataset using loc.
5. Concatenate the subsets for fraud and non-fraud datasets.
6. Shuffle the entire datasets.
7. Write the two subsets to separate CSV files.

4.3 Algorithm

4.3.1 Logistic Regression

- i. Define the hypothesis function as:

$$h(x) = 1/(1 + e^{-z})$$

where z is the linear combination of the features and their corresponding weights:

$$z = w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n$$

and $w_0, w_1, w_2, \dots, w_n$ are the weights for each feature and x_1, x_2, \dots, x_n are the feature values.

- ii. Define the cost function as the average of the log-loss over all training examples:

$$j(w) = (-1/m) * \sum (y_i * \log(h(x_i)) + (1-y_i) * \log(1 - h(x_i)))$$

Where m is the number of training examples, y_i is the true label (0 or 1).

- iii. Initialize the weights randomly, such as $w_0, w_1, w_2, \dots, w_n$.
- iv. Set the learning rate (α) and the number of iterations (n) for gradient descent.
- v. For each iteration from 1 to n :

- a. Calculate the predicted values using the hypothesis function:

$$h_i = h(x_i)$$

- b. Calculate the gradient of the cost function with respect to each weight:

$$dw_j = (1/m) * \sum ((h_i - y_i) * x_{ij}) \text{ for } j \text{ in } 0, 1, \dots, n$$

- c. Update the weights using the gradient and the learning rate:

$$w_j = w_j - \alpha * dw_j \text{ for } j \text{ in } 0, 1, \dots, n$$

- vi. After the iterations are completed, the final weights can be used to make predictions on new examples. The predicted value is obtained by passing the feature vector through the hypothesis function and rounding off the result to the nearest integer:
 $y_{\text{pred}} = \text{roundoff}(h(x))$

4.3.2 Boosting Model (XGBoost)

- i. Model is initialized. This model usually is set to the average of the target variable or any other simple model.

$$F_0(x) = \arg_{\gamma} \min \sum_{i=1}^n L(y_i, \gamma)$$

hyperparameters can be set as:

- learning_rate
- n_estimators
- subsample
- subsample
- colsample_bytree
- min_samples_split
- min_samples_left

- min_child_weight
- max_depth
- max_leaf_nodes
- max_features
- loss
- init
- random_state
- verbose
- warm_start
- presort

ii. For $m = 1$ to M :

a. Compute the residual errors:

$$r_{im} = - \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x) = F_{m-1}(x)} \quad \text{for } i = 1, 2, \dots, n.$$

b. Fit a base model $h_m(x)$ to the residual errors. This base model is decision tree in our project. But it can also be a regression tree or any other machine learning algorithm that can handle regression tasks.

It is trained using the training set $\{(x_i, x_{im})\}$ where $i = 1$ to n

c. Compute the step size γ_m that minimizes the following loss function:

$$\gamma_m = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, f_{m-1}(x_i) + \gamma h_m(x_i))$$

Where, $L(y_i, f_{m-1}(x_i) + \gamma h_m(x_i))$ is a loss function that measures the differences between the predicted value and actual value. The loss function can be Mean Squared Error (MSE), Mean Absolute Error (MAE), Huber Loss or any other suitable loss function.

d. Update the model b adding the base model weighted by the step size:

$$f_m(x) = f_{m-1}(x) + \gamma_m h_m(x)$$

iii. The final model is updated by calculating the sum of the initial model and the weighted sum of the base models:

$$f(x) = f_0(x) + \sum_{m=1}^M \gamma_m h_m(x)$$

4.3.3 Genetic Algorithm

1. Hyperparameters of GBM are randomly selected as an initial population.
Population = $\{p_1, p_2, p_3 \dots, p_n\}$, where p_1, p_2, \dots, p_n are the possible set of parameters for the GBM.
2. The fitness of each candidate set of hyperparameters in the population based on a fitness function is evaluated that measures how well each set of hyperparameters performs in the gradient boosting model. The fitness function could be performance matrix of GBM such as accuracy, AUC, F1 score etc.
 $\text{fitness}(p_i) = \text{evaluate_gb_model}(p_i)$
3. The best-performing candidate solutions from the population is selected to create a new population for the next generation.
 $p' = \{p'_1, p'_2, \dots, p'_n\}$, where each p'_i is a selected candidate solution from the population based on their fitness value.
4. New candidate solutions are created by combining the genetic material of two or more selected solutions through crossover.
 $p_i' = \text{crossover}(p_{i1}, p_{i2})$
5. Random changes to the genetic material of some candidate solutions are introduced to maintain diversity.
 $p_i'' = \text{mutate}(p_i')$
6. Some of the worst performing candidates are replaced with the newly created candidate solutions.
 $\text{population}' = \{p_1, p_2, \dots, p_k, p_i'', \dots, p_n\}$, where k is the number of candidate solutions in the new population.
7. Steps 2-6 are repeated until a termination criterion is met.

CHAPTER 5

5. Result analysis and Discussions

As explained above, two algorithms are being used for the dataset which are compared in order to measure the performance of the credit card fraud detection project. To compare the performance between the two algorithms, six measures are being utilized: Accuracy, Precision, Recall, F1 Score, Confusion Matrix, and AUC-ROC Curve.

5.1 Performance Analysis of Logistic Regression

Unbalanced Dataset

accuracy of train data is: 0.9988500954596327

accuracy of test data is: 0.9991397773954567

confusion_matrix :

```
[[56870  8]
```

```
[ 41  43]]
```

accuracy_score : 0.9991397773954567

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	1.00	1.00	1.00	56878
---	------	------	------	-------

1	0.84	0.51	0.64	84
---	------	------	------	----

accuracy			1.00	56962
----------	--	--	------	-------

macro avg	0.92	0.76	0.82	56962
-----------	------	------	------	-------

weighted avg	1.00	1.00	1.00	56962
--------------	------	------	------	-------

AUC-ROC curve

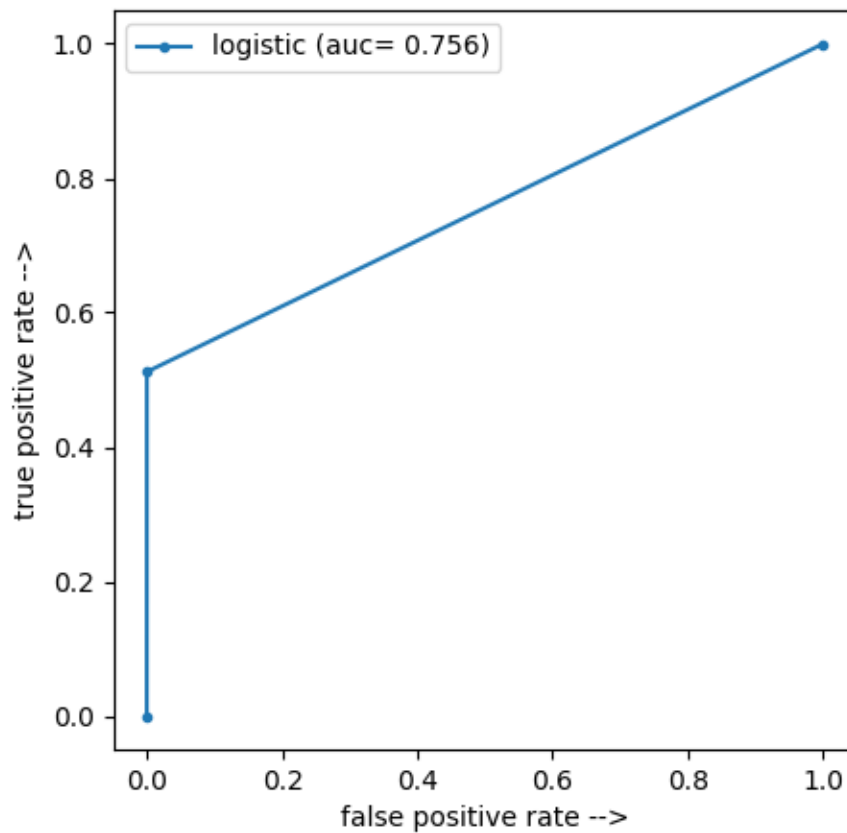


Figure 7 : Unbalanced dataset AUC-ROC curve

Balanced Dataset (using Library)

accuracy of train data is: 0.9097839898348158

accuracy of test data is: 0.9137055837563451

confusion_matrix :

[[95 0]

[17 85]]

accuracy_score : 0.9137055837563451

precision recall f1-score support

0	0.85	1.00	0.92	95
1	1.00	0.83	0.91	102

accuracy		0.91	197
macro avg	0.92	0.92	0.91 197
weighted avg	0.93	0.91	0.91 197

AUC-ROC curve

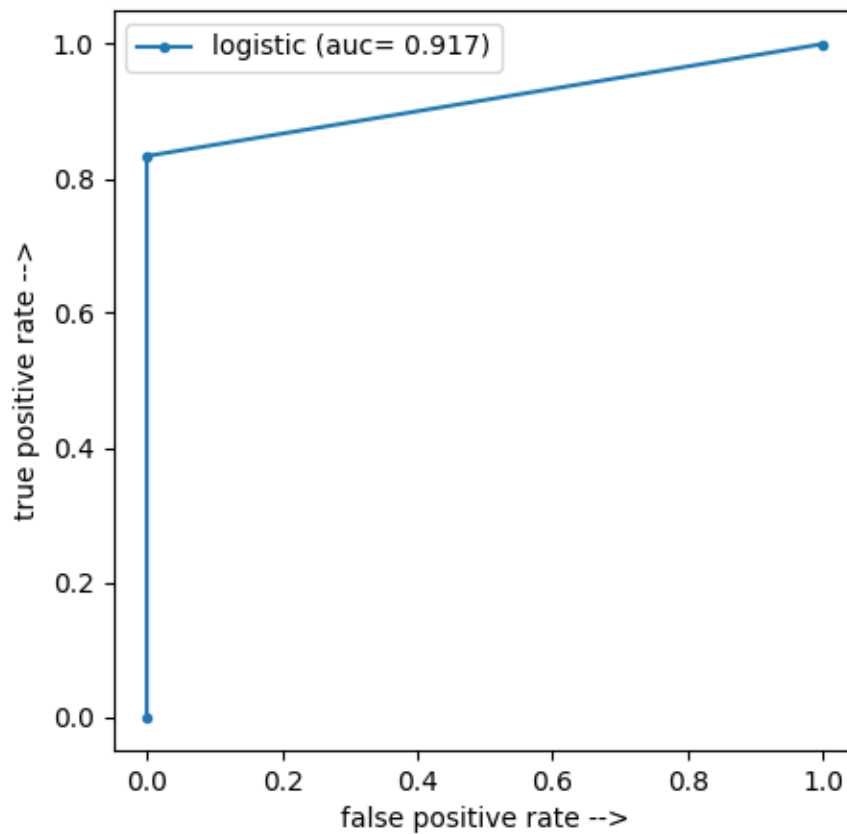


Figure 8: Balanced AUC-ROC curve

Balanced Datasets (from scratch without using Library)

accuracy of train data is: 0.9097839898348158

accuracy of test data is: 0.9137055837563451

confusion_matrix :

```
[[95  0]
```

```
 [17 85]]
```

accuracy_score : 0.9137055837563451

	precision	recall	f1-score	support
0	0.85	1.00	0.92	95
1	1.00	0.83	0.91	102
accuracy			0.91	197
macro avg	0.92	0.92	0.91	197
weighted avg	0.93	0.91	0.91	197

5.2 Performance Analysis of Boosting Model

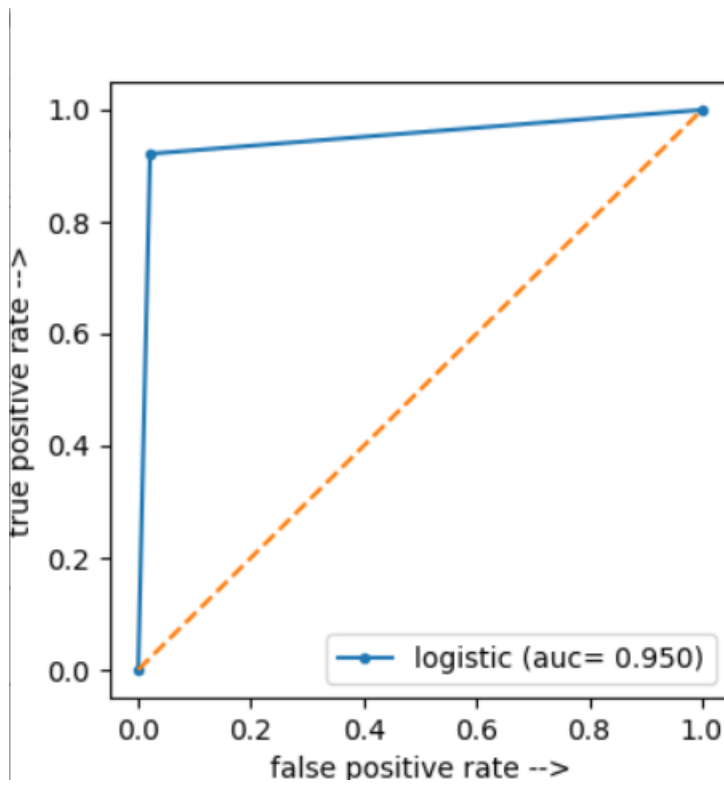


Figure 9: Boosting Model AUC- ROC curve

5.3 Comparison of different models on different datasets

For Dataset “creditcard1.csv”

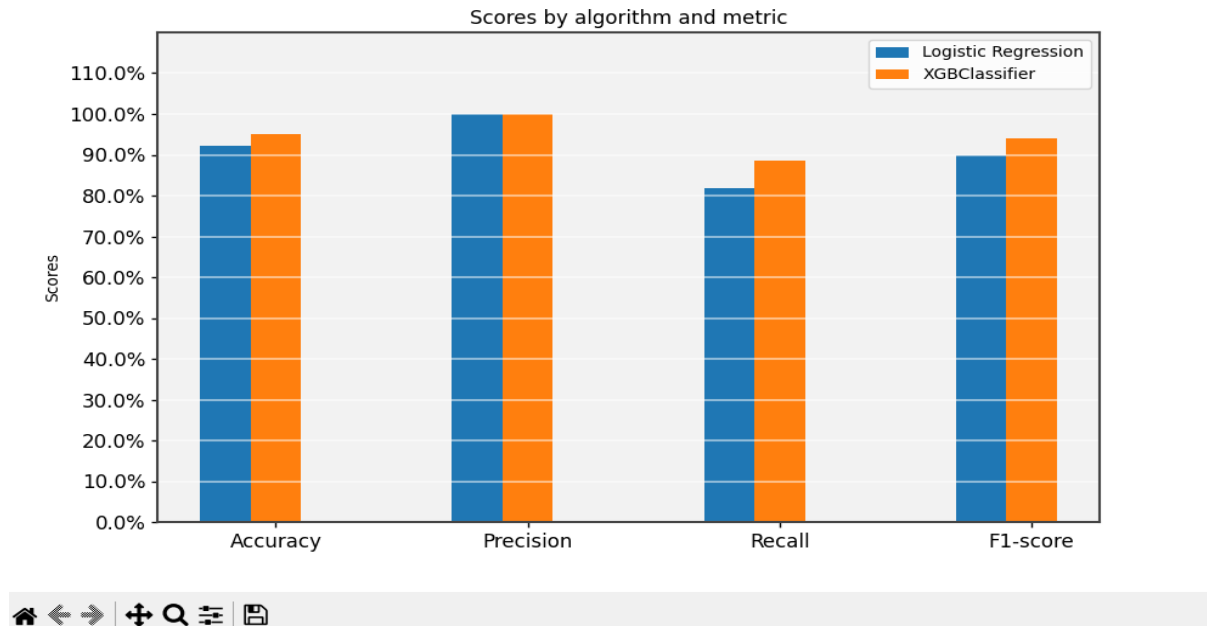


Figure 10: Bar chart of creditcard1.csv

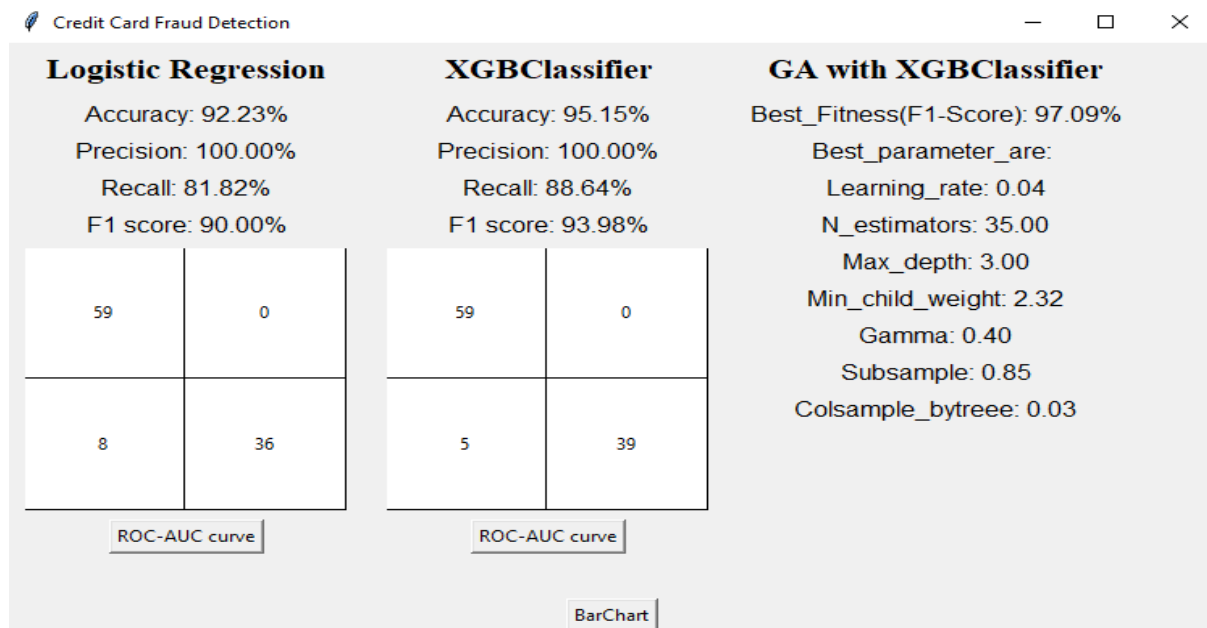


Figure 11: Result of creditcard1.csv

For Dataset “creditcard2.csv”

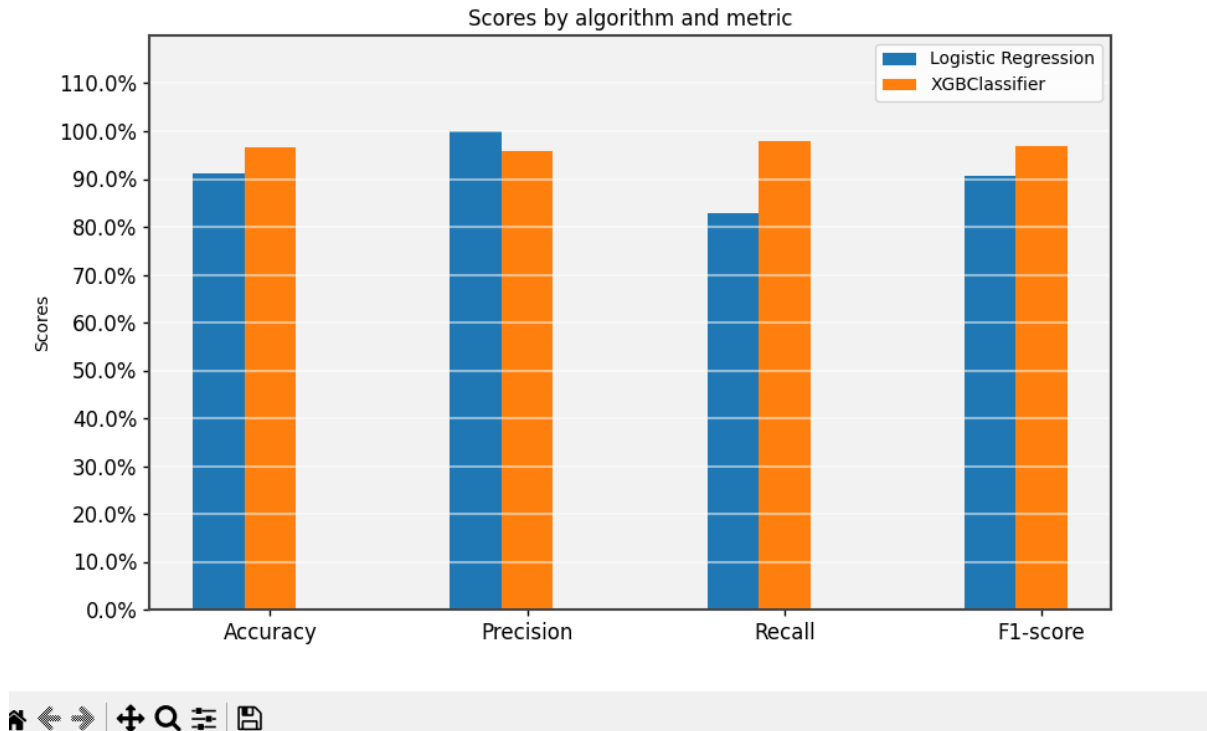


Figure 12: Bar chart of creditcard2.csv

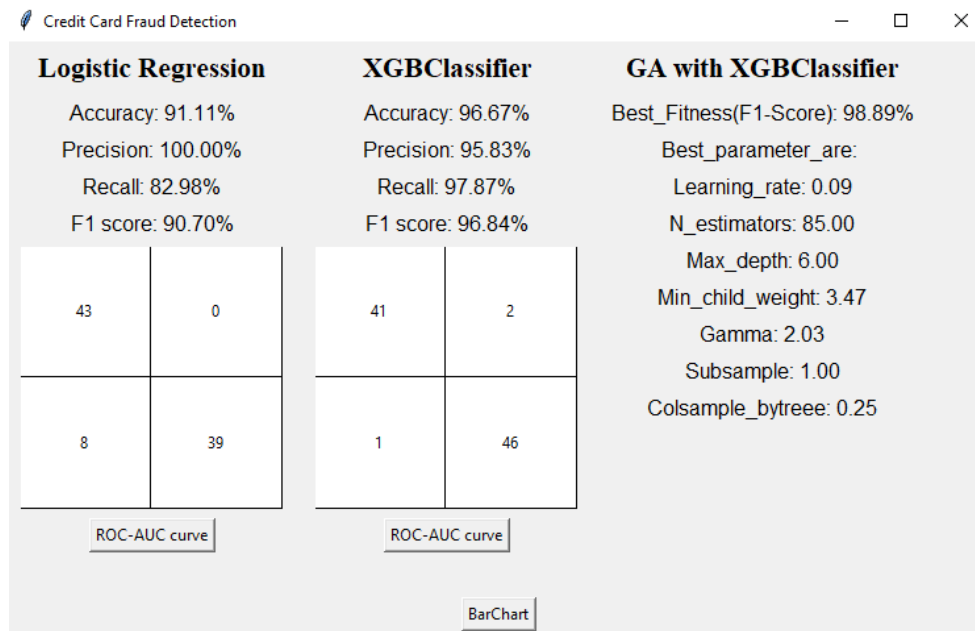


Figure 13: Result of credicard2.csv

When comparing Logistic Regression and XGClassifier algorithms based on Accuracy, Precision, Recall, and F1-Score, Logistic Regression performed better in terms of Precision, while XGClassifier had good results in terms of Accuracy, Recall, and F1-Score.

The Genetic Algorithm was used with XGClassifier, and the results were compared with two other algorithms. It was found that the Genetic Algorithm with XGClassifier had the best performance in terms of F1-Score. The Genetic Algorithm was employed with XGClassifier to optimize the performance of it.

CHAPTER 6

6. Conclusion and Future recommendation

6.1 Conclusion

Based on the comparative analysis of various machine learning algorithms for credit card fraud detection, it was found that unbalanced datasets can result in overly accurate data, indicating the possibility of overfitting. Additionally, the performance of the algorithms depends on factors such as dataset size, complexity of fraud patterns, and hyperparameter tuning.

It was observed that XGBoost generally outperformed Logistic Regression in all aspects except Precision. Furthermore, the Genetic Algorithm was used to optimize the performance of XGBoost, which resulted in the best overall results with fitness score 98%.

These findings highlight the importance of carefully selecting the appropriate algorithm and tuning its hyperparameters for credit card fraud detection. By doing so, credit card companies and financial institutions can enhance their fraud detection systems and reduce the financial losses caused by fraud.

6.2 Future Works

In future, in order to validate the effectiveness of models and parameter tuning approach, other credit card fraud detection datasets can be used to see how well they generalize incorporating with our algorithms into your analysis. It is important to note that in future, the XGBoost model can also be developed from scratch without relying on the library. The more advance hyperparameters techniques such as Bayesian Optimization or Gradient-based Optimization can be explored to see if they lead to better results. Different feature engineering approaches can also be investigated to see how they impact to the models.

References

- [1] *Credit card fraud detection using supervised machine learning methods*, 2022.
- [2] R. S. Faroque Ahmed, "A Comparative Study of Credit Card Fraud Detection Using the Combination of Machine Learning Techniques with Data Imbalance Solution," in *IEEE*, Stanford, CA, 2021.
- [3] S. K. R, "Comparative Analysis of Credit Card Fraud Detection using Machine Learning and Deep Learning Techniques," *International Research Journal of Engineering and Technology (IRJET)*, vol. 08, no. 08, p. 1149, 2021.
- [4] E. H. Alkhamash, "An Optimized Gradient Boosting Model by Genetic Algorithm," *MDPI*, p. 13, 2022.
- [5] K. W. K. Y. P. Y. XinYing Chew, "Credit Card Fraud Detection Using a New Hybrid Machine Learning Architecture," *MDPI*, vol. 10, no. 9, 2022.
- [6] P. ., G. S. ., R. S. S. K. Ratna Sree Valli, "Credit card fraud detection using Machine learning algorithms," *Journal of Research in Humanities and Social Science*, vol. 8, no. 2, pp. 04-11, 2020.
- [7] S. Vats, "Genetic algorithms for credit card fraud detection," in *Conference: International Conference on Education and Educational Technologies*, Sector-7 GIDA, Gorakhpur, 2013.
- [8] D. ., B. ., J. R. Siva Subramanian, "Customer Analysis Using Machine Learning Algorithms: A Case Study Using Banking Consumer Dataset," authors and IOS press, Coimbatore, India, 2021.
- [9] A. R. a. A. T. Mehryar Mohri, *Foundations of Machine Learning*, Cambridge, Massachusetts and London, England: MIT Press.
- [10] I. S. ., N. P. ., P. H. Kasarapu Ramani, "Gradient Boosting Techniques for Credit Card fraud detection," *Journal of Algebraic Statistics*, vol. 13, no. 03, pp. 553-558, 2022 June

Appendices

Screenshot of datasets

Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	
0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.991390	-0.311169	1.46
0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.489095	-0.143772	0.63
1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.717293	-0.165946	2.34
1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.507757	-0.287924	-0.63
2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.345852	-1.119670	0.17

Figure 14: Datasets part1

V15	V16	V17	V18	V19	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
8177	-0.470401	0.207971	0.025791	0.403993	0.251412	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
3558	0.463917	-0.114805	-0.183361	-0.145783	-0.069083	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
15865	-2.890083	1.109969	-0.121359	-2.261857	0.524980	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
31418	-1.059647	-0.684093	1.965775	-1.232622	-0.208038	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
75121	-0.451449	-0.237033	-0.038195	0.803487	0.408542	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

Figure 15: Datasets part2

Before balancing

```
In [6]: sns.countplot(x='class', data = df)
```

```
Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x2270767c188>
```

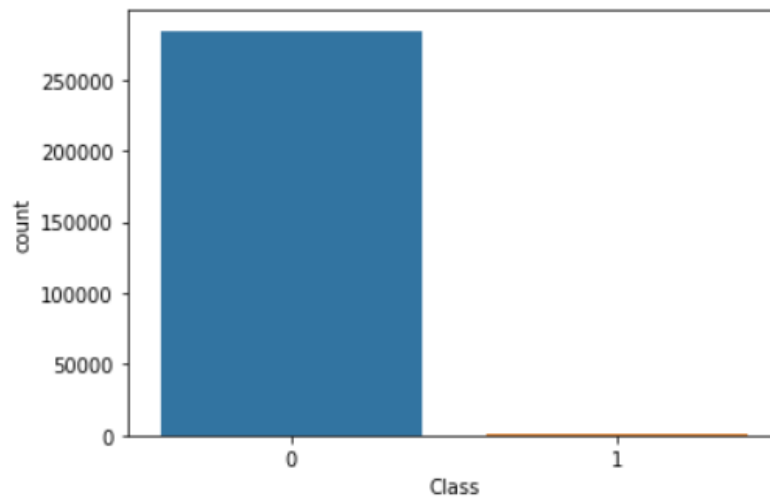


Figure 16: Imbalance data

After balancing

```
In [61]: sns.countplot(x='Class', data = ndf)
```

```
Out[61]: <matplotlib.axes._subplots.AxesSubplot at 0x2270b94d4c8>
```

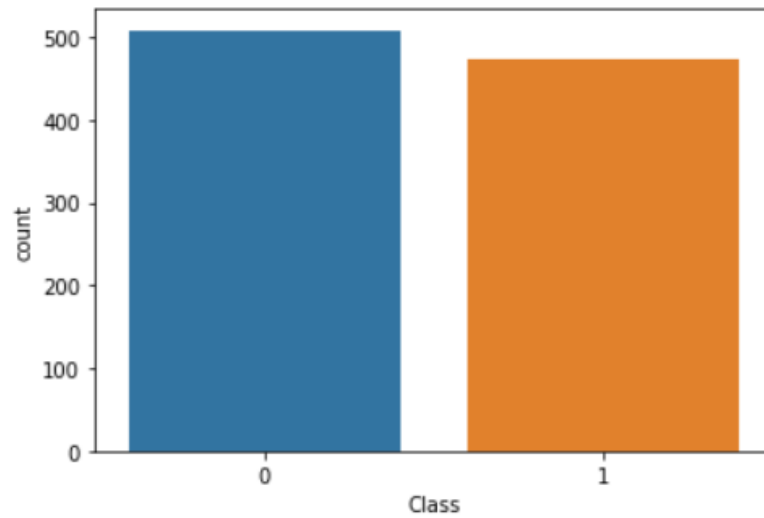


Figure 17: Balanced data

Source code of algorithm used in project

Logistic Regression Algorithm

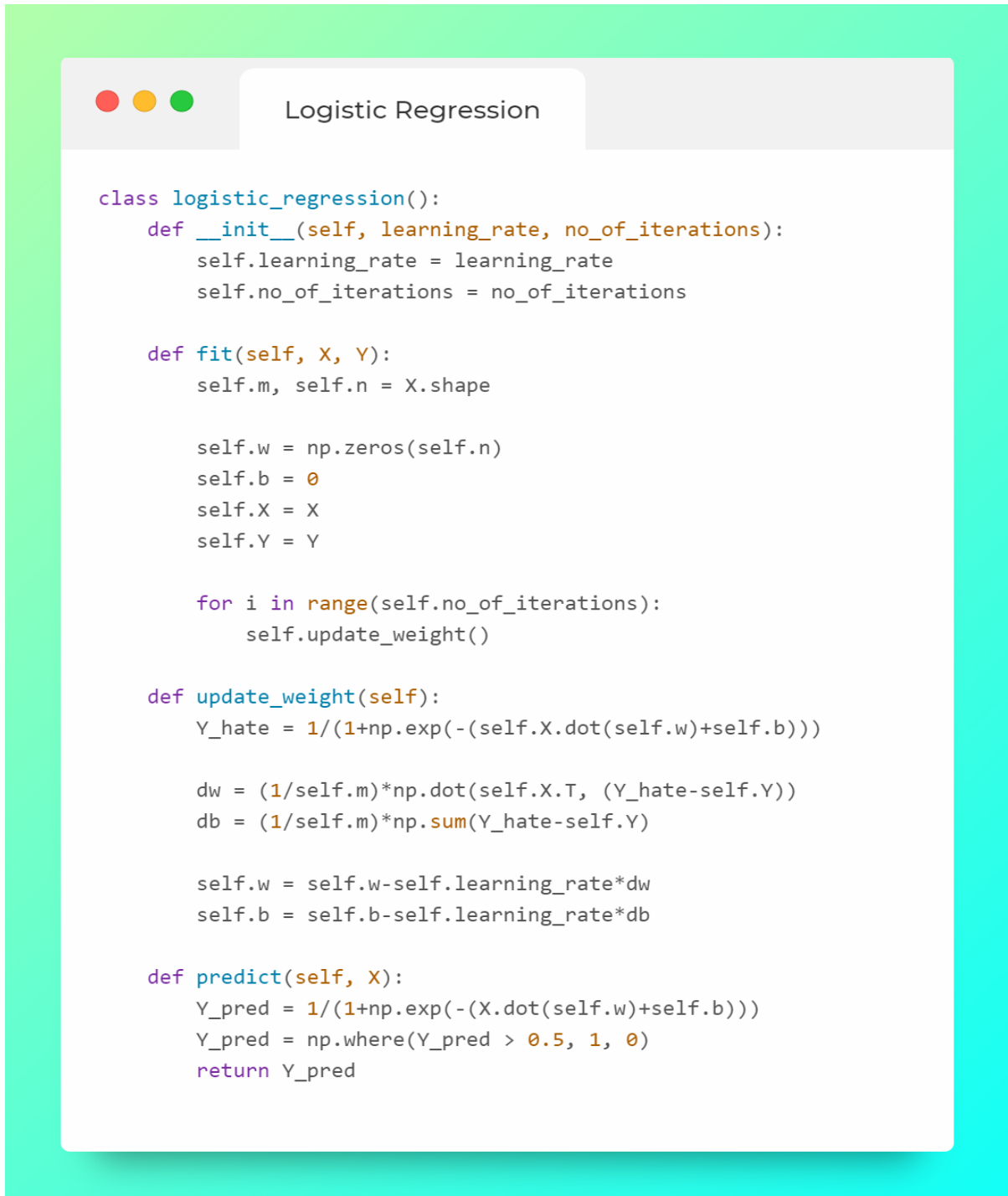

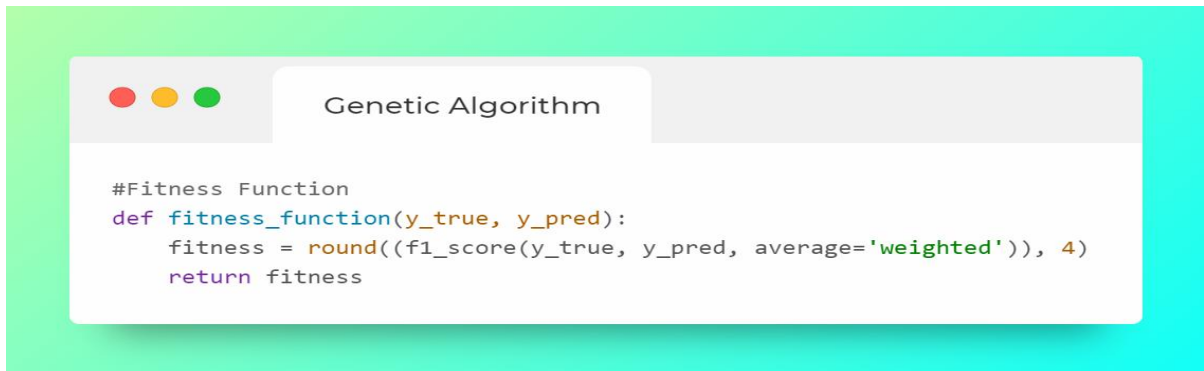


Figure 18: Logistic regression algorithm code



```
#Initialization
def initialize_population(numberOfParents):
    learningRate = np.empty([numberOfParents, 1])
    nEstimators = np.empty([numberOfParents, 1])
    maxDepth = np.empty([numberOfParents, 1])
    minChildWeight = np.empty([numberOfParents, 1])
    gammaValue = np.empty([numberOfParents, 1])
    subSample = np.empty([numberOfParents, 1])
    colSampleByTree = np.empty([numberOfParents, 1])
    for i in range(numberOfParents):
        print(i)
        learningRate[i] = round(random.uniform(0.01, 1), 2)
        nEstimators[i] = random.randrange(10, 150, step = 25)
        maxDepth[i] = int(random.randrange(1, 10, step= 1))
        minChildWeight[i] = round(random.uniform(0.01, 10.0), 2)
        gammaValue[i] = round(random.uniform(0.01, 10.0), 2)
        subSample[i] = round(random.uniform(0.01, 1.0), 2)
        colSampleByTree[i] = round(random.uniform(0.01, 1.0), 2)
    population = np.concatenate((learningRate, nEstimators,
    maxDepth, minChildWeight, gammaValue, subSample,
    colSampleByTree), axis= 1)
    return population
```

Figure 19: Genetic algorithm initialization code



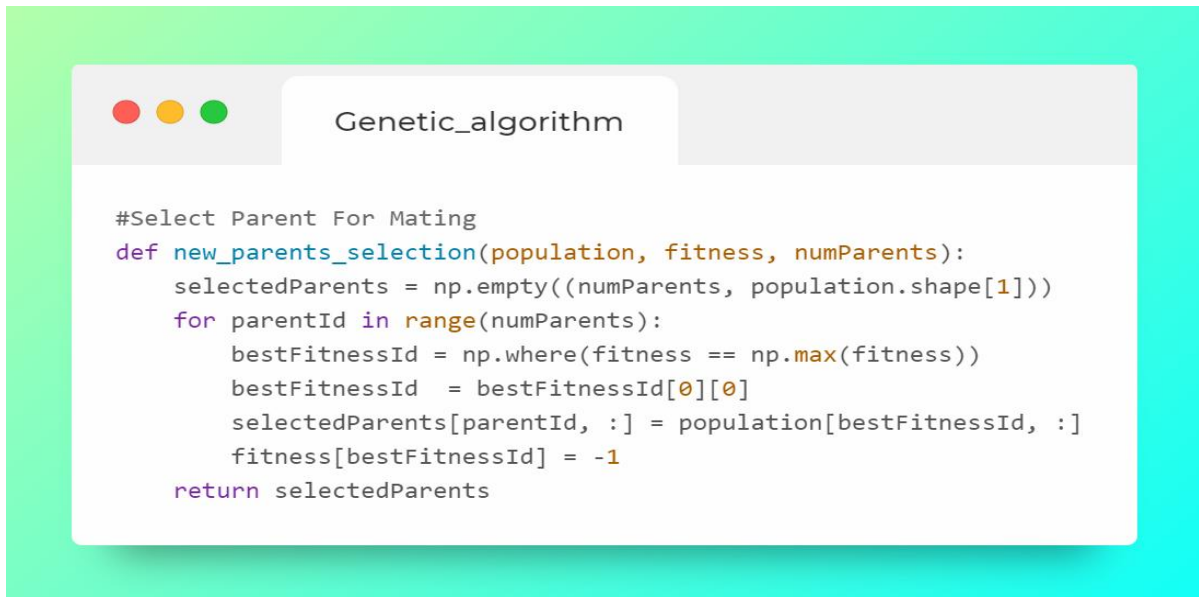
```
#Fitness Function
def fitness_function(y_true, y_pred):
    fitness = round((f1_score(y_true, y_pred, average='weighted')), 4)
    return fitness
```

Figure 20: Genetic algorithm fitness function code



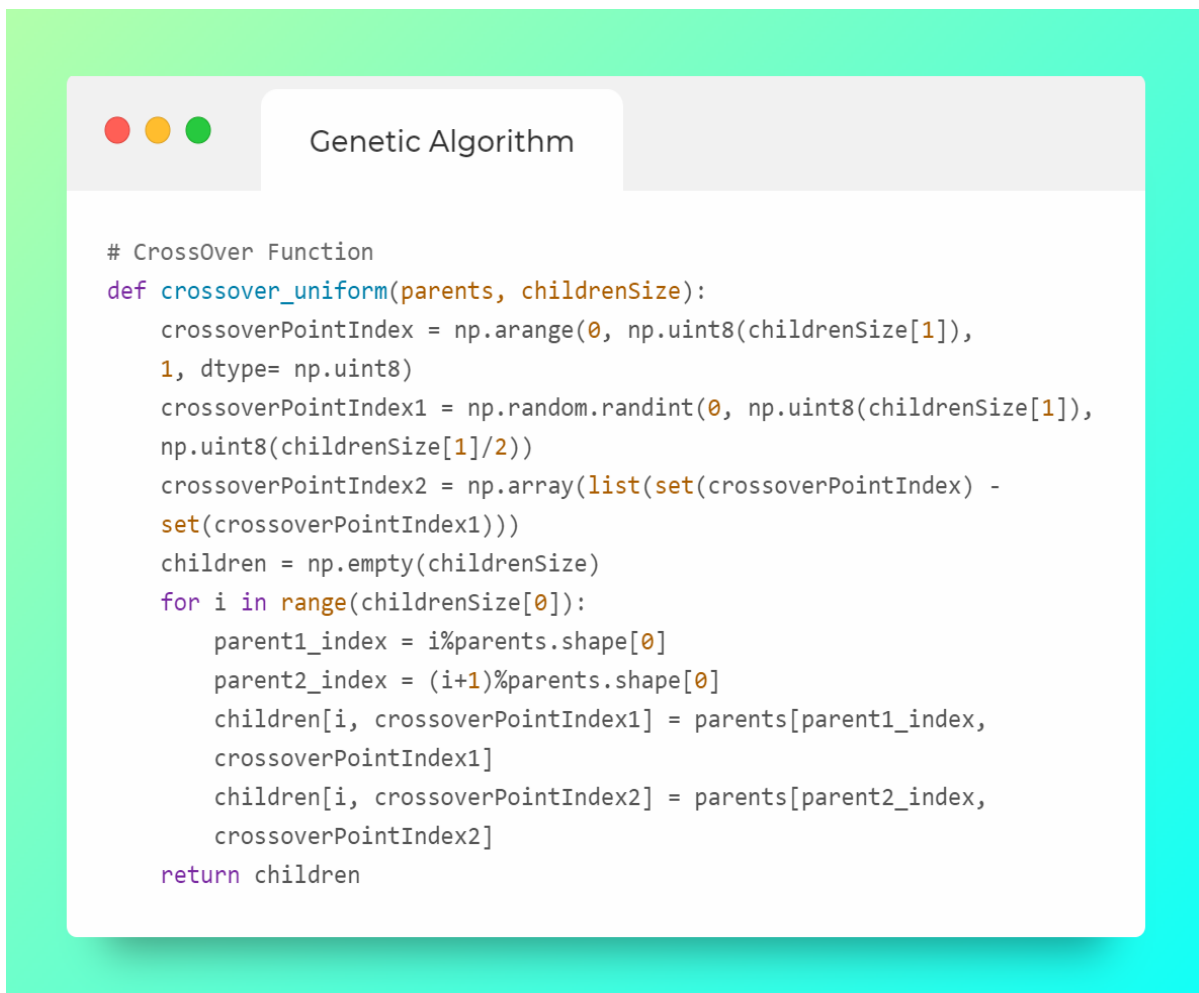
```
# Train Population
def train_population(population, X_train, y_train, X_test, y_test):
    fitness_score = []
    for i in range(population.shape[0]):
        param = { 'objective': 'binary:logistic',
                  'learning_rate': population[i][0],
                  'n_estimators': int(population[i][1]),
                  'max_depth': int(population[i][2]),
                  'min_child_weight': population[i][3],
                  'gamma': population[i][4],
                  'subsample': population[i][5],
                  'colsample_bytree': population[i][6],
                  'seed': 24}
        XGBC = XGBClassifier(n_jobs = 4, random_state = 123).set_params(**param)
        model = XGBC.fit(X_train, y_train)
        preds = model.predict(X_test)
        fitness_score.append(fitness_function(y_test, preds))
    return fitness_score
```

Figure 21: Genetic algorithm train population code



```
#Select Parent For Mating
def new_parents_selection(population, fitness, numParents):
    selectedParents = np.empty((numParents, population.shape[1]))
    for parentId in range(numParents):
        bestFitnessId = np.where(fitness == np.max(fitness))
        bestFitnessId = bestFitnessId[0][0]
        selectedParents[parentId, :] = population[bestFitnessId, :]
        fitness[bestFitnessId] = -1
    return selectedParents
```

Figure 22: Genetic algorithm parent selection code



```
# CrossOver Function
def crossover_uniform(parents, childrenSize):
    crossoverPointIndex = np.arange(0, np.uint8(childrenSize[1]),
    1, dtype= np.uint8)
    crossoverPointIndex1 = np.random.randint(0, np.uint8(childrenSize[1]),
    np.uint8(childrenSize[1]/2))
    crossoverPointIndex2 = np.array(list(set(crossoverPointIndex) -
    set(crossoverPointIndex1)))
    children = np.empty(childrenSize)
    for i in range(childrenSize[0]):
        parent1_index = i%parents.shape[0]
        parent2_index = (i+1)%parents.shape[0]
        children[i, crossoverPointIndex1] = parents[parent1_index,
        crossoverPointIndex1]
        children[i, crossoverPointIndex2] = parents[parent2_index,
        crossoverPointIndex2]
    return children
```

Figure 23: Genetic algorithm crossover uniform code

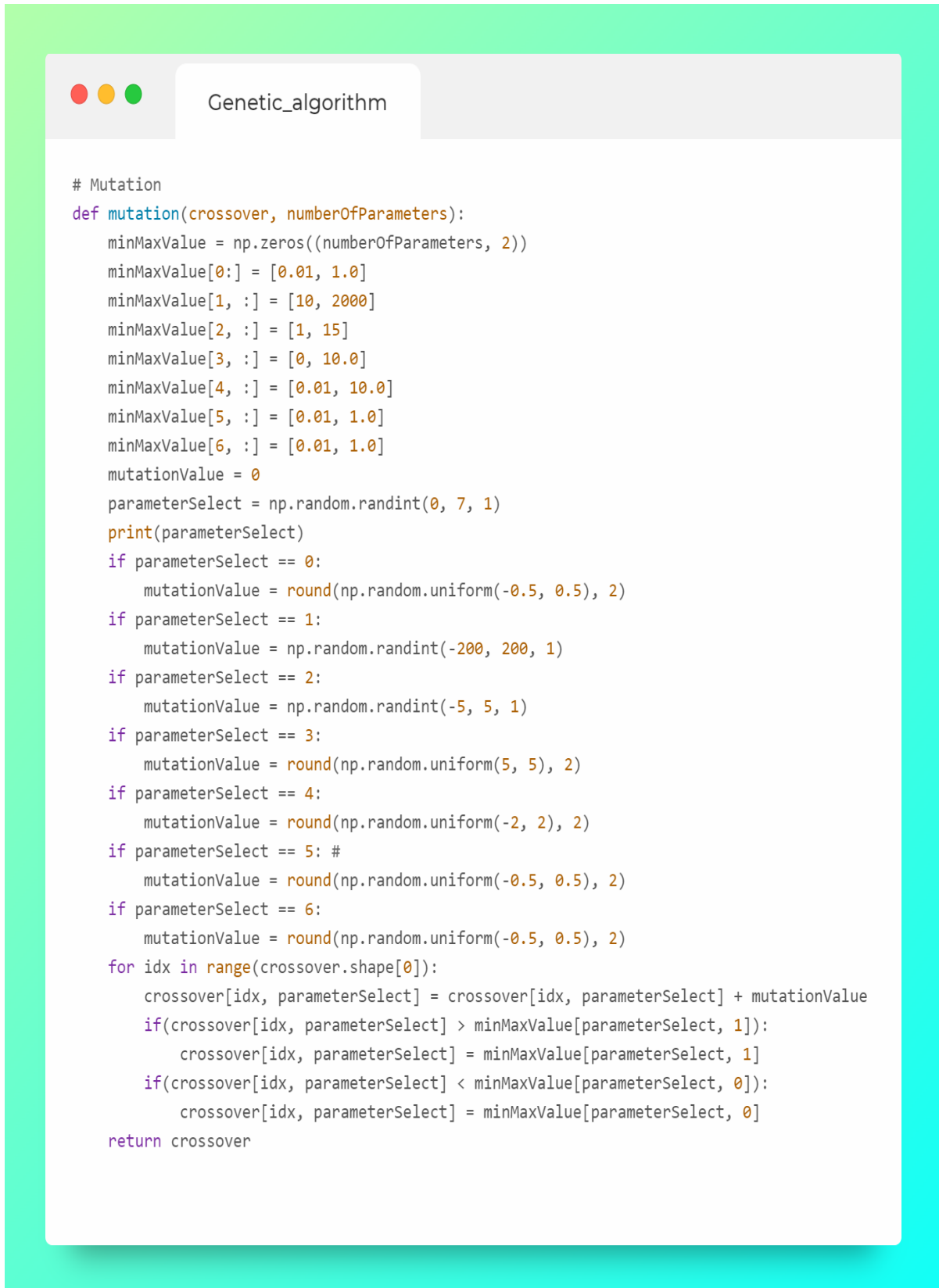


Figure 24: Genetic algorithm mutation code

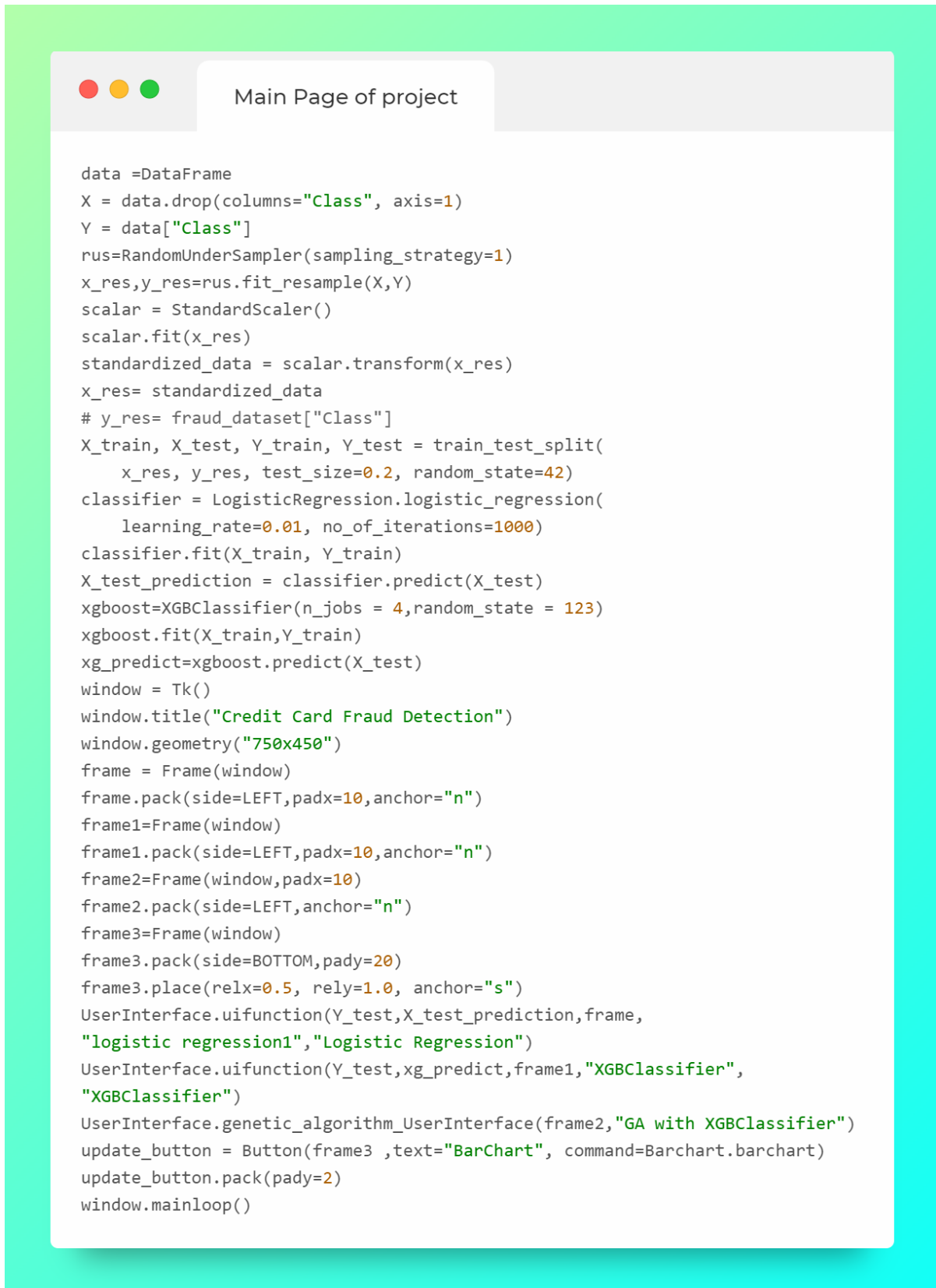


Figure 25: Main Page code