Detecting Fraudulent Transactions with Ensemble and Conventional Models in Highly Imbalanced Large-Scale Datasets*

Ngoc Nguyen Duy[1]
nguyenduyngoccter@gmail.com
Student ID: 21522380

Lan Nguyen Thi Thanh[2]
emeraldlan1105@gmail.com
Student ID: 21521065

Thinh Le Quang[3]
21522635@gm.uit.edu.vn
Student ID: 21522635

Linh Bui Le Khanh[4]
linhbui8103@gmail.com
Student ID: 21522284

***Abstract*—Fraudulent transaction detection in dynamic urban environments presents a complex challenge due to various interconnected factors. This research focuses on the comparative evaluation of machine learning models for fraud detection, including ensemble methods (XGBoost, CatBoost) and conventional algorithms (Decision Trees, Support Vector Machines and K-Nearest Neighbors). Evaluation methods such as the Precision-Recall (PR) AUC, F1 Score, F2 Score are employed to assess model performance. The study aims to uncover patterns in fraudulent transaction detection, providing valuable insights to strengthen the integrity of financial systems. The results show that the ensemble methods, specifically XGBoost and CatBoost, outperformed the conventional algorithms in terms of fraud detection. These models achieved higher Precision-Recall (PR) AUC, F1 Score, and F2 Score, indicating their superior performance in identifying fraudulent transactions. On the other hand, the conventional algorithms (KNN, Decision Tree, SVM) showed slightly better results in predicting non-fraudulent transactions.**

***Index Terms*—Fraudulent Transactions, Ensemble, Boosting, Decision Trees, Support Vector Machines, K-Nearest Neighbors, Precision-Recall (PR) AUC, F1 Score, F2 Score**

I. Introduction

Fraudulent transaction detection in dynamic urban environments presents a significant challenge due to its complex and interconnected nature. To combat the increasing sophistication and diversity of fraudulent activities, robust and efficient detection methods are needed. Machine learning algorithms have emerged as powerful tools for fraud detection, capable of uncovering hidden patterns and anomalies in large volumes of transactional data.

This research aims to compare and evaluate various machine learning models for fraud detection, including ensemble methods like XGBoost and CatBoost, known for their ability to harness the collective intelligence of multiple models. conventional algorithms such as Decision Trees, Support Vector Machines (SVM), and K-Nearest Neighbors (KNN) are also examined to assess their individual performance.

Evaluation metrics such as Precision-Recall (PR) AUC, F1 Score, and F2 Score are used to measure the effectiveness of the models in detecting fraudulent transactions. These metrics provide insights into the models' ability to accurately identify fraud cases, balance precision and recall, and emphasize the importance of recall in fraud detection scenarios.

The research aims to uncover patterns and insights in fraudulent transaction detection, contributing to the enhancement of financial system integrity and security. By understanding the strengths and weaknesses of different machine learning models, organizations can adopt effective fraud detection strategies tailored to their specific needs. The study also provides insights into the potential of ensemble methods and conventional algorithms, aiding future advancements in fraud detection techniques.

Preliminary results indicate that ensemble methods, particularly XGBoost and CatBoost, demonstrate superior performance in fraud detection tasks. These models exhibit higher Precision-Recall AUC, F1 Score, and F2 Score, indicating their ability to accurately identify fraudulent transactions. conventional algorithms (KNN, Decision Tree, SVM) show slightly better results in predicting non-fraudulent transactions. These findings contribute to the knowledge on effective fraud detection methodologies and can guide the implementation of robust fraud prevention measures.

In the subsequent sections of this research paper, we provide an in-depth analysis of the datasets used, the methodology for model training and evaluation, and detailed results and discussions. By examining these aspects, we aim to offer a comprehensive understanding of the comparative performance of various machine learning models for fraud detection, benefiting both academic researchers and industry practitioners.

II. Related Work

In the field of fraudulent transaction detection, multiple studies have employed machine learning techniques to improve prediction accuracy. Some relevant works are discussed below:

Chen et al. (2018) conducted a study on fraudulent credit card transaction detection using XGBoost algorithm. They compared the performance of XGBoost with other popular machine learning algorithms and found that XGBoost outperformed the others, achieving high accuracy in identifying fraudulent transactions.

*This is our report for CS116 - Python for Machine Learning (Spring 2023) at University of Information Technology - Vietnam National University HCMC

[1]Faculty of Computer Science, University of Information Technology - Vietnam National University HCMC

Li et al. (2019) explored the application of CatBoost algorithm in detecting fraudulent online transactions. Their results demonstrated that CatBoost showed promising performance in terms of precision, recall, and F1 score, indicating its potential as an effective classifier for fraud detection.

These studies provide a clear indication of the viability and success of ensemble machine learning algorithms in identifying fraudulent transactions. Nevertheless, given the large, imbalanced dataset presented in our study, we aim to investigate the performance of both ensemble and conventional machine learning algorithms, aiming to develop a more efficient and accurate model for fraud transaction detection.

III. DATA UNDERSTANDING AND PRE-PROCESSING

A. Data Description

The dataset utilized in this research study, obtained from Kaggle, is specifically designed for fraud detection. It comprises a comprehensive collection of financial transaction records within a company. The dataset encompasses a vast amount of information, with a total of 6,353,307 samples.

The dataset contains various features that provide valuable insights into the nature of the financial transactions. These features are carefully selected to facilitate the identification and analysis of fraudulent activities. A detailed description of the dataset features is presented below:

1) *Step* - The number of minutes elapsed between each transaction.
2) *Type* - The type of transaction, which can be one of the following categories: CASH_IN, CASH_OUT, DEBIT, PAYMENT, TRANSFER.
3) *Amount* - The amount of money involved in the transaction.
4) *nameOrig* - The name of the person or entity originating the transaction.
5) *oldbalanceOrg* - The original balance of the remittance account before the transaction took place.
6) *newbalanceOrig* - The balance of the remittance account after the transaction was performed.
7) *nameDest* - The name of the person or entity receiving the transaction.
8) *oldbalanceDest* - The original balance of the recipient's account before the transaction took place.
9) *newbalanceDest* - The balance of the recipient's account after the transaction was performed.
10) *isFraud* - Identifies a fraudulent transaction (1) and non fraudulent (0).

The objective of this dataset is to identify fraudulent financial transactions. The data is used to train machine learning models capable of accurately detecting fraudulent activities, thereby enhancing the financial security measures.

B. Data understanding and basic EDA

In the data understanding and basic exploratory data analysis (EDA) phase, our goal is to gain insights into the original dataset and identify any patterns or trends that may exist. We will examine the features of the dataset and conduct exploratory analysis to extract useful observations. This step is crucial in understanding the data distribution, identifying potential outliers or missing values, and determining the relationships between variables. By conducting EDA, we can lay the groundwork for further analysis and model development in the subsequent steps.

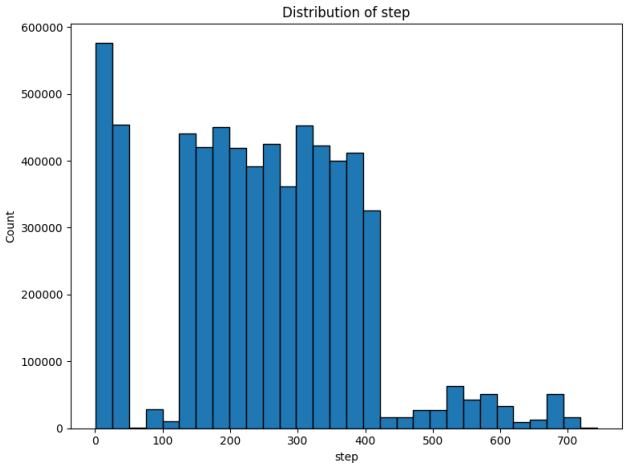

Fig. 1. Distribution of step

It can be seen that the number of minutes that a transaction takes to complete is concentrated in the range of 120 - 420 minutes.

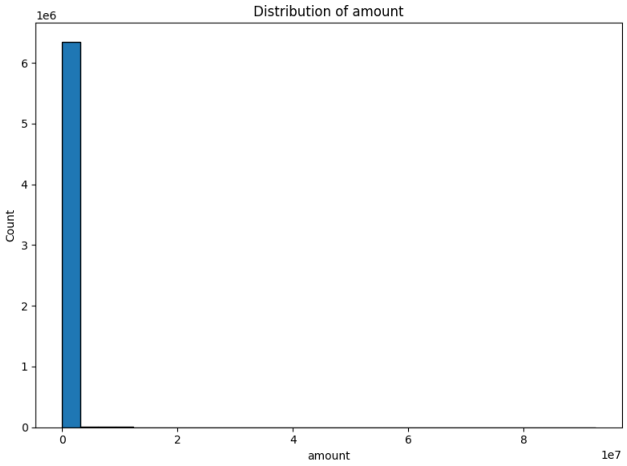

Fig. 2. Distribution of amount

It can be observed that the amount in each transaction is distributed very evenly, mostly in the range of less than 10 million.

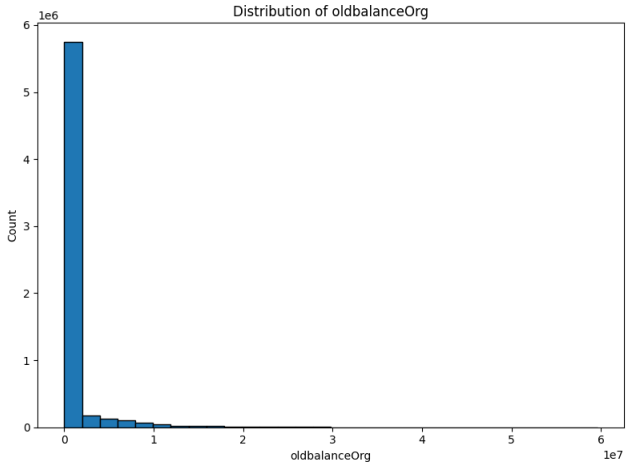

Fig. 3. Distribution of oldbalanceOrg

It can be seen that the initial balance in the account of the remitter in each transaction is only distributed in the range of less than 20 million, and mainly from 0 to 2 million.

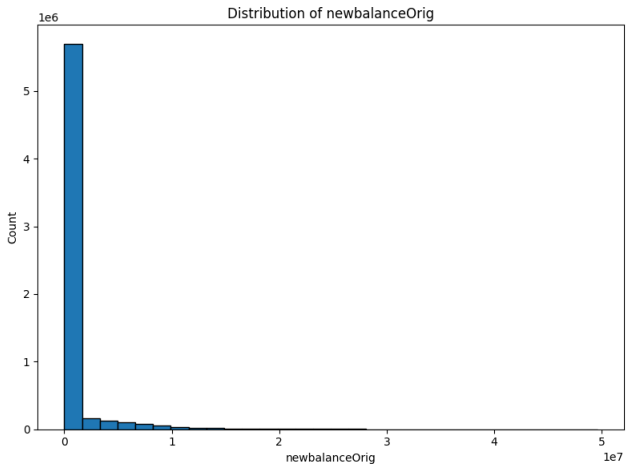

Fig. 4. Distribution of newbalanceOrg

For the new balance after the transaction, there is a similar comment.

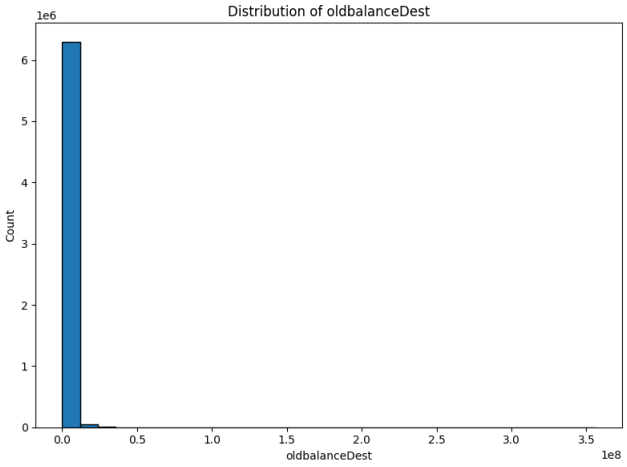

Fig. 5. Distribution of oldbalanceDest

It can be seen that the initial balance in the recipient's account in each transaction is distributed in the range of less than 50 million, and mainly from 0 to 20 million.

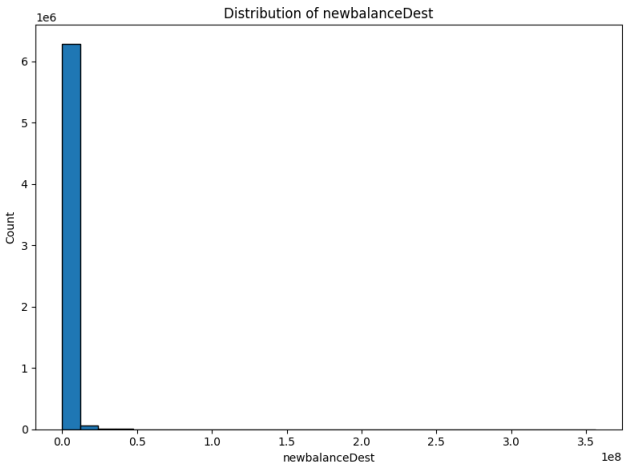

Fig. 6. Distribution of newbalanceDest

And we also have the same observation with the new balance of the recipient's account after the transaction.

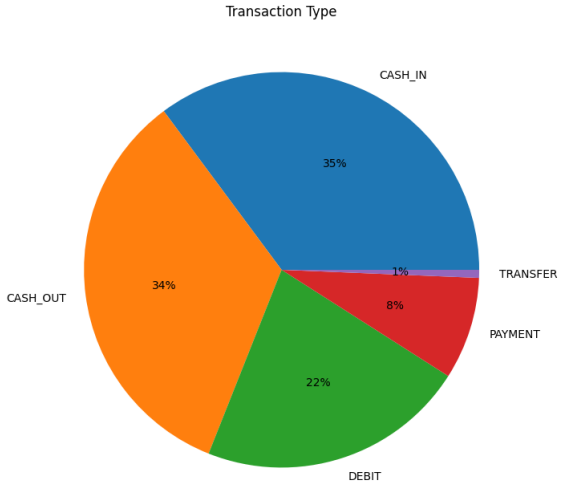

Fig. 7. Distribution of transaction type

It is observed that, transaction type CASH_IN accounts for the largest proportion (35%) and TRANSFER accounts for the smallest proportion (1%). The transactions mainly fall into the CASH_IN, CASH_OUT and DEBIT.

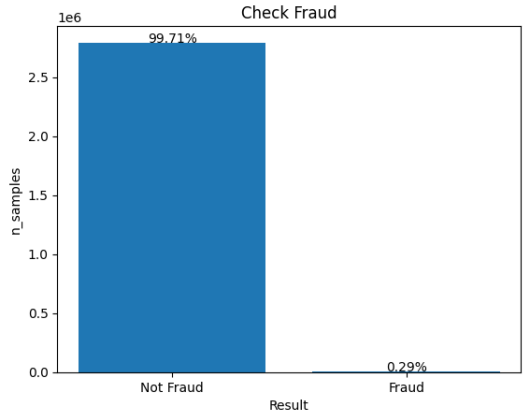

Fig. 8. Distribution of isFraud

The ratio of fraudulent transactions to the total number of transactions is very low (0.29%). This shows that the data is not balanced (imbalenced), with a very small number of fraudulent transactions compared to the number of non-fraudulent transactions. This is also consistent with reality, because the number of fraudulent transactions usually accounts for a very small proportion

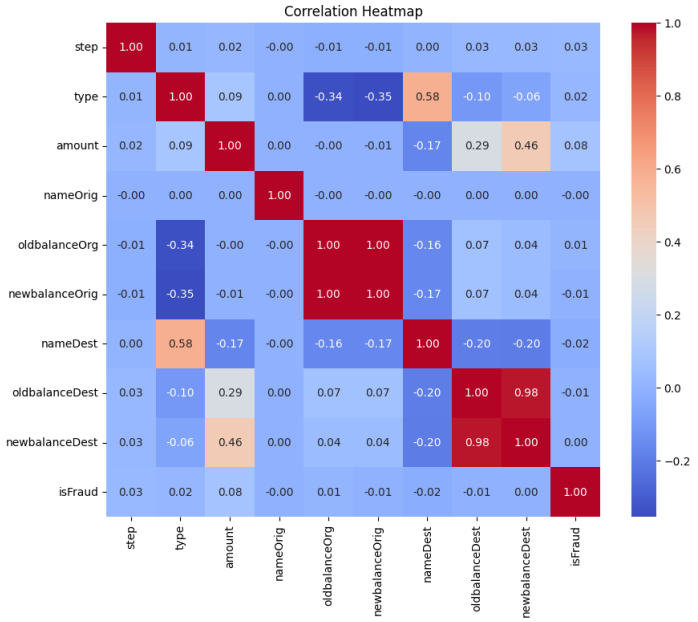

Fig. 9. The correlation matrix of the dataset

As observed, the isFraud column has no special interest in the remaining columns, in other words, no column has a great influence on the result of the isFraud column.

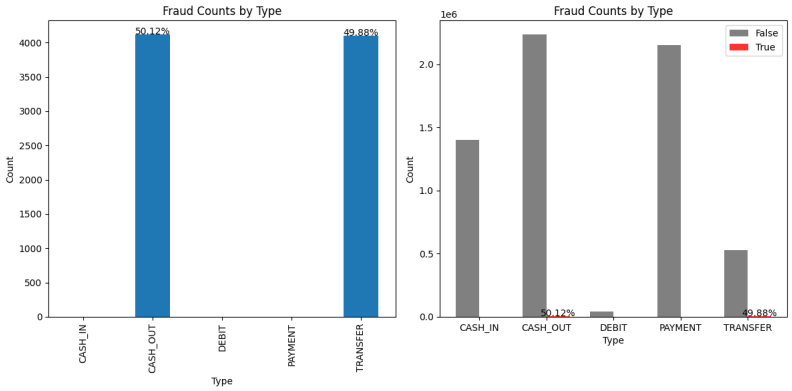

Fig. 10. Statistics of fraudulent transactions based on transaction type

After analyzing the fraudulent transactions based on transaction type, we made an interesting observation: fraudulent transactions are only present in two groups, namely CASH_OUT (50.12%) and TRANSFER (49.88%). The other three transaction types, namely CASH_IN, PAYMENT, and DEBIT, do not contain any fraudulent transactions.

To reduce the initial large amount of data, we decided to keep only 10,000 samples for each of the three transaction types (CASH_IN, PAYMENT, and DEBIT). This allows us to focus on a subset of the data while still retaining representative samples for these transaction types.

C. Data Preprocessing

To preprocess the data, two important methods are employed: LabelEncoder and StandardScaler.

1) LabelEncoder

The LabelEncoder technique is utilized to transform categorical variables into numerical values. This step is particularly valuable when working with machine learning algorithms that necessitate numerical inputs.

The LabelEncoder assigns a unique numerical label to each distinct category in the categorical variable. This ensures that the algorithm can effectively process the data without relying on the categorical nature of the variable.

2) StandardScaler

The StandardScaler method is employed to standardize the numerical features of the dataset. Standardization involves transforming the values of each feature to have a mean of zero and a standard deviation of one.

This step is crucial when the numerical features possess varying scales or units. Standardizing the features helps bring them to a comparable scale, enhancing the performance of certain machine learning algorithms that are sensitive to feature scale.

By utilizing StandardScaler, the features are transformed to have a mean of zero and a standard deviation of one.

These preprocessing steps ensure that the data is in an appropriate format for further analysis and model training.

IV. OUR METHODS

A. XGBoost

XGBoost (eXtreme Gradient Boosting) is a powerful and widely used machine learning algorithm known for its exceptional performance in various tasks, including classification and regression problems. It is an ensemble learning method that combines the predictions of multiple individual models (weak learners) to create a strong predictive model.

XGBoost operates by constructing a series of decision trees sequentially, where each subsequent tree attempts to correct the mistakes made by the previous trees. The algorithm optimizes a specific objective function by iteratively adding trees to the ensemble.

XGBoost in classification problems - XGBoost can be applied to classification tasks by adapting the algorithm to handle discrete class labels. It uses a technique called gradient boosting to build an ensemble of decision trees that collectively make predictions.

The key steps involved in the XGBoost algorithm are as follows:

- Initialization: Assign initial scores or predictions to all data points.
- Constructing trees: Build decision trees iteratively, with each tree attempting to minimize the overall loss function of the ensemble.
- Gradient descent optimization: Calculate the gradients of the loss function and update the scores or predictions of each data point based on the gradients.
- Regularization: Apply regularization techniques such as controlling the depth of trees, adding penalties to the loss function, or subsampling the training data to prevent overfitting and improve generalization.
- Ensemble prediction: Combine the predictions of all the individual trees to obtain the final classification result.

a) Determining the objective function

In XGBoost, the choice of the objective function is crucial as it defines the optimization problem that the algorithm aims to solve. The objective function quantifies the quality of the model's predictions and guides the construction of individual trees.

The most commonly used objective functions in XGBoost for classification problems include:

Logistic Loss: This objective function is suitable for binary classification problems. It models the probability of the positive class using the logistic function, and the objective is to maximize the likelihood of the observed labels.

$$\text{Objective} = \sum_{i=1}^{n} \left[y_i \log(1 + e^{-\hat{y_i}}) + (1 - y_i) \log(1 + e^{\hat{y_i}}) \right]$$

where y_i represents the true label of the i-th example, and $\hat{y_i}$ denotes the predicted score.

Softmax Loss: This objective function extends the logistic loss to multi-class classification problems. It models the probabilities of all classes using the softmax function, and the objective is to maximize the likelihood of the observed labels across all classes.

$$\text{Objective} = -\sum_{i=1}^{n} \sum_{j=1}^{K} y_{ij} \log \left(\frac{e^{\hat{y_{ij}}}}{\sum_{k=1}^{K} e^{\hat{y_{ik}}}} \right)$$

where y_{ij} represents the indicator function for the true label of the i-th example belonging to class j, and $\hat{y_{ij}}$ denotes the predicted score for class j.

b) Determining the hyperparameters

XGBoost also includes various hyperparameters that need to be tuned to achieve optimal performance. Some important hyperparameters include:

Learning Rate: Controls the step size at each boosting iteration. A lower learning rate makes the algorithm converge slower but may yield better results.

Maximum Depth: Specifies the maximum depth of each decision tree. Deeper trees can capture more complex interactions but may overfit the training data.

Number of Trees: Determines the number of decision trees in the ensemble. Increasing the number of trees can improve performance, but there is a trade-off with computation time.

Subsample Ratio: Specifies the fraction of the training data to be used for each tree. It helps prevent overfitting and can speed up training.

Regularization Parameters: These parameters control the amount of regularization applied to prevent overfitting. They include parameters for L1 and L2 regularization on leaf weights and the minimum loss reduction required to partition a leaf node.

c) Mathematical Formulation of XGBoost

To understand the inner workings of XGBoost, let's delve into its mathematical formulation. In XGBoost, the goal is to build an ensemble of decision trees that collectively make accurate predictions. The final prediction for a given input is obtained by summing the predictions of all individual trees.

Let's denote the training dataset as $(x_i, y_i)_{i=1}^{n}$, where x_i represents the input features and y_i represents the corresponding true labels. The objective is to find a set of K decision trees, $f_k(x)$, where $k = 1, 2, ..., K$, that minimize a specific loss function.

The overall prediction for an input x can be represented as:

$$\hat{y} = \sum_{k=1}^{K} f_k(x)$$

The XGBoost algorithm optimizes the objective function by iteratively adding decision trees to the ensemble. At each iteration, a new tree is constructed to correct the mistakes made by the existing ensemble. This process is guided by gradient descent optimization.

The objective function for XGBoost can be written as:

$$\text{Objective} = \sum_{i=1}^{n} \ell(y_i, \hat{y}i) + \sum k = 1^{K} \Omega(f_k)$$

where $\ell(y_i, \hat{y}_i)$ is the loss function that measures the discrepancy between the true label y_i and the predicted value $\hat{y}_i$. The second term, $\Omega(f_k)$, represents the regularization term that penalizes complex trees to control overfitting.

During each iteration, a new decision tree is added to the ensemble by minimizing the objective function. The tree construction process involves recursively partitioning the data based on specific splitting criteria, such as information gain or gain in the objective function. The optimal splits are determined through an exhaustive search.

To prevent overfitting, XGBoost employs regularization techniques such as pruning, which prunes the tree structure by removing unnecessary splits. Additionally, the regularization terms in the objective function, such as L1 or L2 regularization, control the complexity of individual trees.

The optimization of the objective function is performed using gradient descent. The gradients of the loss function with respect to the predictions are computed and used to update the predictions at each iteration. This process aims to find the optimal predictions that minimize the overall loss.

By iteratively adding trees, updating predictions, and optimizing the objective function, XGBoost gradually improves its ensemble's predictive performance. The hyperparameters mentioned earlier, such as learning rate, maximum depth, and regularization parameters, play crucial roles in this optimization process and should be carefully tuned to achieve the best results.

B. CatBoost

CatBoost is a gradient boosting algorithm designed to handle categorical features in classification and regression tasks. It stands for "Category Boosting" and is known for its ability to automatically handle categorical variables without the need for extensive preprocessing.

CatBoost in classification problems - CatBoost can effectively handle categorical features by using a variant of gradient boosting that incorporates an innovative approach called ordered boosting. This approach uses an optimal ordering of categorical variables during the training process, which helps capture dependencies and interactions between categories.

The key steps involved in the CatBoost algorithm are as follows:

- Handling categorical features: CatBoost applies various techniques to handle categorical variables, such as converting them into numerical representations based on statistics or using an algorithm called "random permutation" to encode the categories.
- Initialization: Assign initial scores or predictions to all data points.
- Constructing trees: Build decision trees iteratively, with each tree attempting to minimize the overall loss function of the ensemble. CatBoost uses a gradient-based algorithm that considers the gradients of the loss function for both numerical and categorical features.
- Regularization: Apply regularization techniques to prevent overfitting, such as constraining the tree structure, feature sampling, and using learning rate schedules.
- Ordered boosting: CatBoost uses an optimal ordering of categorical features during the boosting process to capture dependencies between categories more effectively.
- Ensemble prediction: Combine the predictions of all the individual trees to obtain the final classification result.

CatBoost provides several advantages, including robust handling of categorical features, automatic feature scaling, and support for GPU acceleration. It also offers built-in tools for hyperparameter tuning and model interpretation.

By leveraging its innovative techniques for handling categorical variables and utilizing gradient boosting, CatBoost excels in various machine learning tasks, particularly those involving complex datasets with mixed data types.

a) Determining the objective function

In CatBoost, the choice of the objective function plays a critical role in defining the optimization problem and guiding the training process. The objective function quantifies the quality of the model's predictions and shapes the construction of individual trees.

For classification problems, CatBoost offers various objective functions, including:

Logloss (Cross-Entropy): This objective function is commonly used for binary classification problems. It measures the

logarithmic loss between the predicted probabilities and the true binary labels.

$$\text{Objective} = -\sum_{i=1}^{n} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

where y_i represents the true binary label of the i-th example, and p_i denotes the predicted probability of the positive class.

MultiClass (Softmax): For multi-class classification problems, CatBoost employs the softmax objective function. It models the probabilities of all classes using the softmax function and maximizes the likelihood of the observed labels across all classes.

$$\text{Objective} = -\sum_{i=1}^{n} \sum_{j=1}^{K} y_{ij} \log(p_{ij})$$

where y_{ij} represents the indicator function for the true label of the i-th example belonging to class j, and p_{ij} denotes the predicted probability of class j.

CatBoost also offers additional objective functions and customizability to address specific classification tasks and requirements.

The choice of the objective function depends on the nature of the problem and the desired behavior of the model. It is important to select an appropriate objective function that aligns with the evaluation metric and the underlying problem's characteristics.

By selecting the right objective function, CatBoost can effectively optimize the model's predictions and enhance its performance in classification tasks.

b) Determining the hyperparameters

CatBoost provides a range of hyperparameters that can be tuned to optimize the model's performance. These hyperparameters include:

Learning Rate: Similar to XGBoost, the learning rate in CatBoost controls the step size at each boosting iteration. It determines the impact of each tree on the final prediction and affects the convergence speed of the algorithm.

Maximum Depth: The maximum depth parameter specifies the maximum depth of each decision tree in the ensemble. Deeper trees can capture more complex interactions but may increase the risk of overfitting.

Number of Trees: CatBoost also allows you to specify the number of trees in the ensemble. Increasing the number of trees can improve model performance, but there is a trade-off with computational resources.

Subsample Ratio: The subsample ratio hyperparameter determines the fraction of the training data to be used for each tree. It can help prevent overfitting by introducing randomness and reducing the correlation between trees.

Regularization Parameters: CatBoost offers regularization techniques to control overfitting. These include L1 and L2 regularization on leaf weights, which penalize large weight values, and the minimum loss reduction required to partition a leaf node.

In addition to these hyperparameters, CatBoost provides several other options for customization, such as handling categorical features, feature importance calculation methods, and early stopping criteria.

To determine the optimal hyperparameter values, techniques like grid search, random search, or Bayesian optimization can be applied. These methods involve systematically exploring the hyperparameter space to find the combination that maximizes the model's performance on a validation set.

By appropriately tuning the hyperparameters, CatBoost can be fine-tuned to achieve better generalization and improved performance in various classification tasks.

c) Mathematical Formulation of CatBoost

To gain insight into the inner workings of CatBoost, let's explore its mathematical formulation. CatBoost aims to build an ensemble of decision trees that collectively make accurate predictions. The final prediction for an input is obtained by summing the predictions of all individual trees.

Consider a training dataset $(x_i, y_i)_{i=1}^{n}$, where x_i represents the input features and y_i represents the corresponding true labels. The objective is to find a set of K decision trees, $f_k(x)$, where $k = 1, 2, ..., K$, that minimize a specific loss function.

The overall prediction for an input x can be represented as:

$$\hat{y} = \sum_{k=1}^{K} f_k(x)$$

The CatBoost algorithm optimizes the objective function by iteratively adding decision trees to the ensemble. At each iteration, a new tree is constructed to correct the mistakes made by the existing ensemble. This process is guided by gradient descent optimization.

The objective function for CatBoost can be written as:

$$\text{Objective} = \sum_{i=1}^{n} \ell(y_i, \hat{y}i) + \sum k = 1^{K} \Omega(f_k)$$

where $\ell(y_i, \hat{y}_i)$ is the loss function that measures the discrepancy between the true label y_i and the predicted value $\hat{y}_i$. The second term, $\Omega(f_k)$, represents the regularization term that penalizes complex trees to control overfitting.

During each iteration, a new decision tree is added to the ensemble by minimizing the objective function. The tree construction process involves recursively partitioning the data based on specific splitting criteria, such as information gain or gain in the objective function. The optimal splits are determined through an exhaustive search.

To prevent overfitting, CatBoost employs regularization techniques such as pruning, which prunes the tree structure by removing unnecessary splits. Additionally, the regularization terms in the objective function, such as L1 or L2 regularization, control the complexity of individual trees.

The optimization of the objective function is performed using gradient descent. The gradients of the loss function with respect to the predictions are computed and used to update the predictions at each iteration. This process aims to find the optimal predictions that minimize the overall loss.

CatBoost incorporates a unique feature called "ordered boosting" that utilizes the categorical features directly during the tree construction process. This feature preserves the categorical nature of the data and avoids the need for one-hot encoding.

By iteratively adding trees, updating predictions, and optimizing the objective function, CatBoost gradually improves its ensemble's predictive performance. The hyperparameters mentioned earlier, such as learning rate, maximum depth, and regularization parameters, play crucial roles in this optimization process and should be carefully tuned to achieve the best results.

C. Decision Tree

Decision tree is a widely used machine learning algorithm that can be applied to both classification and regression tasks. It creates a flowchart-like structure where each internal node represents a test on a feature, each branch represents the outcome of the test, and each leaf node represents a class label or a numerical value.

Decision tree in classification problems - In classification tasks, a decision tree recursively partitions the feature space based on the values of input features to assign class labels to instances. At each internal node, a decision is made based on a feature's value, which determines the path to follow in the tree. The process continues until a leaf node is reached, which corresponds to the predicted class label.

The key steps involved in building a decision tree are as follows:

- Feature selection: Determine the most informative feature to split the data based on criteria such as information gain, Gini impurity, or variance reduction.
- Splitting: Partition the data based on the selected feature, creating child nodes for each possible outcome of the test.
- Recursive construction: Repeat the splitting process on each child node until a stopping criterion is met. This criterion may include reaching a maximum tree depth, reaching a minimum number of samples in a leaf node, or other pre-defined conditions.
- Leaf node assignment: Assign a class label (in classification) or a numerical value (in regression) to each leaf node based on the majority class or average value of the samples within that node.

Decision trees offer several advantages, including interpretability, ability to handle both categorical and numerical features, and robustness to outliers. However, they can be prone to overfitting if not properly regularized or if the tree depth becomes too large.

To mitigate overfitting, various techniques can be applied, such as pruning the tree, setting a maximum depth, or using regularization parameters. Additionally, ensemble methods like random forests and gradient boosting can be employed to improve the predictive performance of decision trees.

a) Determining the objective function

In decision tree algorithms, the choice of the objective function is crucial as it defines the optimization problem that the algorithm aims to solve. The objective function quantifies the quality of the model's predictions and guides the construction of the decision tree.

For decision trees in classification problems, the most commonly used objective functions include:

Gini Index: The Gini index measures the impurity or disorder of a node in a decision tree. It evaluates the probability of misclassifying a randomly chosen element from the node. The objective is to minimize the Gini index across all nodes of the tree.

Entropy: Entropy is another measure of impurity or disorder. It calculates the information gain by considering the entropy of a node and the weighted average of the entropies of its child nodes. The objective is to minimize entropy and maximize information gain.

By selecting an appropriate objective function, the decision tree algorithm optimizes the predictions and constructs a tree that best fits the training data, leading to accurate predictions on unseen data.

b) Determining the hyperparameters

When using a decision tree algorithm, there are several hyperparameters that can be tuned to optimize the model's performance. These hyperparameters include:

Maximum Depth: The maximum depth of a decision tree limits the number of levels or splits it can have. A deeper tree can capture more complex relationships in the data but may also overfit the training set. Setting an appropriate maximum depth is crucial to balance model complexity and generalization.

Minimum Sample Split: This hyperparameter determines the minimum number of samples required to perform a split at a node. Nodes with fewer samples than the specified threshold will not be further split, which can help prevent overfitting and control tree size.

Minimum Sample Leaf: The minimum number of samples required to be in a leaf node. If a split would result in a leaf node with fewer samples than this threshold, the split is not performed. Similar to the minimum sample split, this parameter helps control the tree size and prevent overfitting.

Split Criterion: The criterion used to measure the quality of a split at each node. Common criteria include Gini impurity and entropy for classification tasks, and mean squared error or mean absolute error for regression tasks. The choice of split criterion depends on the nature of the problem and the desired behavior of the decision tree.

Feature Selection Criterion: The criterion used to evaluate the importance of different features when making splits. Popular criteria include information gain, gain ratio, and Gini importance. The feature selection criterion affects the tree's ability to capture relevant features and can impact its performance.

Pruning Parameters: Pruning is a technique used to reduce the complexity of a decision tree and prevent overfitting. Pruning parameters control the conditions for pruning branches or subtrees based on metrics such as the cost complexity criterion (also known as alpha or complexity parameter).

Pruning can improve generalization by simplifying the tree while maintaining predictive accuracy.

These are just a few examples of hyperparameters that can be tuned in a decision tree algorithm. The optimal values for these hyperparameters depend on the specific dataset and problem at hand. Techniques like grid search, random search, or Bayesian optimization can be employed to systematically search the hyperparameter space and find the combination that maximizes the model's performance on a validation set.

By carefully selecting and tuning the hyperparameters, the decision tree algorithm can be optimized to achieve better predictive performance.

D. K-Nearest Neighbors Algorithm

The k-nearest neighbors algorithm (KNN) is a popular classification method known for its simplicity and effectiveness in handling diverse datasets. Unlike other classifiers, KNN relies on the assumption that similar data points share common properties or belong to the same class.

KNN is a non-parametric, instance-based learning technique that stores the entire training dataset in memory, eliminating the need for a traditional training phase. This unique characteristic categorizes KNN as a "lazy learning" model, where computations occur mainly during classification or prediction.

KNN in classification problems - KNN predicts the class label for a given data point based on the majority vote among its k nearest neighbors. If k is set to 1, the algorithm assigns the data point to the class label of its single nearest neighbor.

The basic steps of the KNN algorithm are:

- Compute the distances between the new data point and all training samples that have already been grouped into clusters.
- Sort the distances in ascending order and choose the k samples with the shortest distance values.
- Apply the voting principle. The new data point will be classified to the largest cluster out of k selected samples.

a) Determining the distance metric

The choice of the distance metric significantly impacts the identification of the nearest neighbors. While there are many different distance measures, this paper addresses the three primary ones: Euclidean distance, Manhattan distance, and Minkowski distance.

Euclidean distance (p=2) is the most commonly used distance measure. It measures the straight-line distance between two points and is widely used for real-valued vectors. In this study, we use Euclidean distance for the calculation of distances.

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Manhattan distance (p=1) is another popular distance metric, also known as taxicab distance or city block distance, which calculates the absolute value of the differences between coordinates, resembling a navigation path through city streets.

$$d(x,y) = \sum_{i=1}^{n}|x_i - y_i|$$

Minkowski distance represents a generalized form of Euclidean and Manhattan distances, where the parameter p enables the creation of additional distance metrics. Notably, a p value of 2 corresponds to Euclidean distance, while a value of 1 represents Manhattan distance.

$$d(x,y) = (\sum_{i=1}^{n}|x_i - y_i|)^{\frac{1}{p}}$$

b) Determining the k value

The k value in KNN determines the number of neighbors considered when classifying an uncategorized sample. The k value needs to be carefully chosen because it directly effects the model's bias-variance trade-off. A lower k value results in higher variance but lower bias, as the decision boundary becomes more sensitive to local fluctuations. Conversely, larger k value tend to lower variance but could bring in more bias, thus oversimplifying the decision boundaries.

Our approach incorporates hyperparameter tuning techniques to determine the optimal k value for enhanced model performance, which we will discuss later in this paper.

E. Support Vector Machine

Support Vector Machine (SVM) are supervised learning algorithms used for classification and regression tasks. They aim to find a hyperplane that maximally separates different classes in the training data. This hyperplane is determined by finding the largest margin, the distance between the hyperplane, and the closest data points from each class. New data can be classified based on which side of the hyperplane it falls on.

Support Vector Machine (SVM) is a relatively simple **Supervised Machine Learning Algorithm** used for classification and/or regression. They belong to a family of generalized linear classifications. A special property of SVM is, SVM simultaneously minimizes the empirical classification error and maximizes the geometric margin. So SVM is called Maximum Margin Classifier. SVM maps the input vector to a higher dimensional space where a maximal separating hyperplane is constructed. Two parallel hyperplanes are constructed on each side of the hyperplane that separates the data. The separating hyperplane is the hyperplane that maximizes the distance between the two parallel hyperplanes. An assumption is made that the larger the margin or distance between these parallel hyperplanes the better the generalization error of the classifier will be

a) Overview of SVM

We consider data points of the form as follows:

$$(x_1, y_1), (x_2, y_2), (x_3, y_3), ..., (x_n, y_n)$$

where $y_n = 1$ or $y_n = -1$ is a constant denoting the class to which that point x_n belongs. n = number of samples. Each x_n is a d-dimensional real vector. The scaling is important to guard against variables (attributes) with larger variance.

We can view this training data, by means of the dividing (or separating) hyperplane, which takes

$$w.x + b = 0$$

where b is the scalar and w is the p-dimensional vector. The vector w points perpendicular to the separating hyperplane. Adding the offset parameter b allows us to increase the margin. Absent of b, the hyperplane is forced to pass through the origin, restricting the solution.

As we are interested in the maximum margin, we are interested in SVM and parallel hyperplanes. Parallel hyperplanes can be described by equation

$$w.x + b = 1$$
$$w.x + b = -1$$

If the training data are linearly separable, we can select these hyperplanes so that there are no points between them and then try to maximize their distance. By geometry, We find the distance between the hyperplane is $2/|w|$. So we want to minimize $|w|$. To excite data points, we need to ensure that for all I either

$$w.x_i – b \geq 1 \quad or \quad w.x_i – b \leq -1 \tag{1}$$

This can be written as:

$$y_i.(w.x_i – b) \geq 1, \quad 1 \leq i \leq n \tag{2}$$

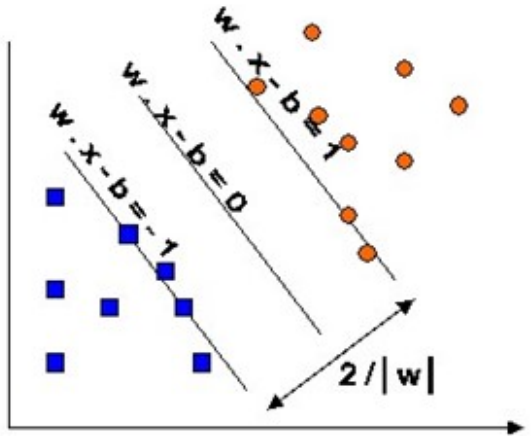

Fig. 11. Maximum margin hyperplanes for a SVM trained with samples from two classes

Samples along the hyperplanes are called Support Vectors (SVs). A separating hyperplane with the largest margin defined by M = $2/|W|$ that is specifies support vectors means training data points closets to it. Which satisfy?

$$y_j[w^T.x_j + b] = 1, i = 1 \tag{3}$$

Optimal Canonical Hyperplane (OCH) is a canonical Hyperplane having a maximum margin. For all the data, OCH should satisfy the following constraints

$$y_i[w^T.x_i + b] \geq 1; i = 1, 2\ldots, n \tag{4}$$

Where l is Number of Training data point. In order to find the optimal separating hyperplane having a maximul margin, A learning macine should minimize $\|W\|^2$ subject to the inequality constraints

$$y_i[w^T.x_i + b] \geq 1; i = 1, 2\ldots, n \tag{5}$$

This optimization problem was solved by the saddle points of Lagrange's Function

$$\begin{aligned} L_p = L_(w, b, a) &= \frac{1}{2}\|w\|^2 - \sum_{i=1}^{N} a_i(y_i(w^T x_i + b) - 1) \\ &= \frac{1}{2}w^T w - \sum_{i=1}^{N} a_i(y_i(w^T x_i + b) - 1) \end{aligned} \tag{6}$$

Where α_i is a Lagranges multiplier. The search for optimal saddle points (w_0, b_0, α_0) is necessary because Lagranges must be minimized with respect to w and b and has to be maximized with respect to nonnegative αi ($\alpha_i \geq 0$). This problem can be solved either in primal form (which is the form of w and b) or in a dual form (which is the form of α_i). Equation numbers (8) and (9) are convex and KKT conditions, which are necessary and sufficient conditions for a maximum of equation (8). Partially differentiate equation () with respect to saddle points (w_0, b_0, α_0).

$$\frac{\delta L}{\delta w_o} = 0 \Rightarrow w_0 = \sum_{i=1}^{N} \alpha_i y_i x_i \tag{7}$$

And

$$\frac{\delta L}{\delta w_0} = 0 \Rightarrow \sum_{i=1}^{N} a_i y_i = 0 \tag{8}$$

Substituting equations (6) and (7) in equation (5). We change the primal form into a dual form.

$$L_d(\alpha) = \sum_{i=1}^{N} -\frac{1}{2}\sum_{i=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j \tag{9}$$

In order to find the optimal hyperplane, a dual lagrangian (L_d) has to be maximized with respect to nonnegative α_i (i .e. α_i must be in the nonnegative quadrant) and with respect to the equality constraints as follow

$$\alpha_i \geq 0 \text{ , i = 1,2.....N}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \tag{10}$$

Note that the dual Lagrangian $L_d(\alpha)$ is expressed in terms of training data and depends only on the scalar products of input patterns ($x_i^T x_j$). More detailed information on SVM can be found in Reference No.[1]&[2].

b) Kernel selection of SVM

Training vectors xi are mapped into a higher (may be infinite) dimensional space by the function Φ. Then SVM finds a linear separating hyperplane with the maximal margin in this higher dimension space. $C > 0$ is the penality parameter of the error term.

Furthermore, $K(x_i, x_j) \equiv \Phi(x_i)^T$ is called the kernel function. There are many kernel functions in SVM, so how to select a good kernel function is also a research issue. However, for general purposes, there are some popular kernel functions :

- Linear kernel: $K(x_i, x_j) = x_i^T x_j$..
- Polynomial kernel: $K(x_i, x_j) = (\gamma x i^T x_j + r)^d, \gamma > 0$
- RBF kernel : $K(x_i, x_j) = exp(-\gamma \left\| x_i - x_j \right\|^2), \gamma > 0$
- Sigmoid kernel: $K(x_i, x_j) = tanh(\gamma x_i^T x_j + r)$

Here, γ, r, and d are kernel parameters. In these popular kernel functions, RBF is the main kernel function because of the following reasons [2]:

1. The RBF kernel nonlinearly maps samples into a higher dimensional space unlike to linear kernel.

2. The RBF kernel has fewer hyperparameters than the polynomial kernel.

3. The RBF kernel has fewer numerical difficulties.

c) Model selection of SVM

Model selection is also an important issue in SVM. Recently, SVM has shown good performance in data classification. Its success depends on the tuning of several parameters which affect the generalization error. We often call this parameter tuning procedure the model selection. If you use the linear SVM, you only need to tune the cost parameter C. Unfortunately, linear SVM are often applied to linearly separable problems. Many problems are non-linearly separable. For example, Satellite data and Shuttle data are not linearly separable. Therefore, we often apply nonlinear kernel to solve classification problems, so we need to select the cost parameter (C) and kernel parameters (γ, d).

We usually use the grid-search method or bayessearch in cross validation to select the best parameter set. Then apply this parameter set to the training dataset and then get the classifier. After that, use the classifier to classify the testing dataset to get the generalization accuracy.

V. EXPERIMENT

A. Dataset

Imbalanced Dataset Challenges - The class imbalance in the financial dataset for fraud detection poses significant challenges for conventional evaluation metrics such as accuracy. In imbalanced datasets, where the negative class (not fraud) is significantly outnumbered by the positive class (fraud), classifiers tend to be biased towards the majority class. As a result, accuracy can be misleading, as a classifier that predicts all instances as non-fraud can still achieve a high accuracy due to the dominance of the negative class. Therefore, alternative metrics are required to assess the performance of classifiers on imbalanced datasets, with a particular focus on correctly identifying the minority class (fraud).

B. Metrics

The evaluation and comparison of different classifiers for fraud detection in financial datasets are essential for building robust and reliable fraud detection systems. This section focuses on precision-related metrics, namely the Precision-Recall Curve, F1 Score, and F2 Score, and their significance in evaluating and comparing classifiers specifically for financial datasets characterized by severe class imbalance.

a) Precision-Recall Curve

The Precision-Recall Curve is a widely used metric for assessing classifier performance on imbalanced datasets. This curve illustrates the tradeoff between precision and recall at different thresholds.

A higher area under the curve (AUC) indicates both high precision and high recall, where high precision corresponds to a low false positive (FP) rate and high recall corresponds to a low false negative (FN) rate. When the classifier achieves high precision and high recall, it indicates accurate predictions and successful identification of the majority of positive instances.

A system with high recall but low precision produces numerous results, but most of them are incorrect compared to the training labels. Conversely, a system with high precision but low recall generates very few results, but the majority of those are correct. An ideal system with high precision and high recall provides a significant number of correctly labeled results.

TABLE I
CONFUSION MATRIX FOR A TWO-CLASS PROBLEM

	Predicted Positive	**Predicted Negative**
Actual Positive	TP	FN
Actual Negative	FP	TN

Precision is calculated as the ratio of true positives to the sum of true positives and false positives,

$$Precision = \frac{TP}{TP + FP}$$

while recall is the ratio of true positives to the sum of true positives and false negatives.

$$Recall = \frac{TP}{TP + FN}$$

Adjusting the threshold can influence precision and recall values. Lowering the threshold may increase precision if all new results are true positives, while it can introduce false positives and decrease precision if the previous threshold was appropriate or too low.

PR-AUC interpretation - Ideally, an algorithm should exhibit both high precision and high recall, but in practice, there is often a trade-off between the two. A good Precision-Recall curve is characterized by a larger area under the curve (AUC). The PR-AUC metric ranges from 0 to 1, with higher values indicating better classifier performance (see Figure 12). A PR-AUC of 1 represents a perfect classifier that achieves high precision and high recall across all thresholds. Conversely, a

PR-AUC of 0.5 indicates a classifier that performs no better than random guessing.

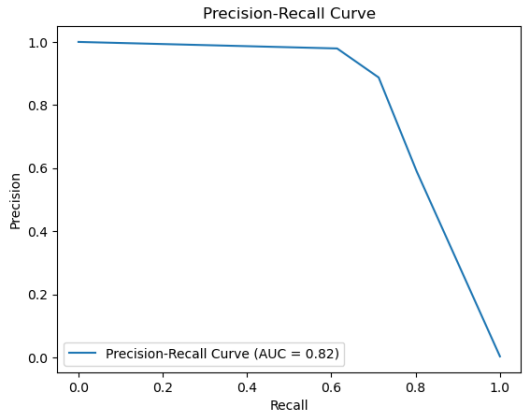

Fig. 12. A Precision-Recall Curve with AUC = 0.82 implies the model has robust classifier performance.

Significance for Fraud Detection - In fraud detection scenarios, prioritizing high precision to minimize false positives is often crucial, as misclassifying a legitimate transaction as fraud can have severe consequences. The PR-AUC metric takes this into account by assigning higher weight to precision. By calculating the area under the Precision-Recall curve, PR-AUC provides an aggregated measure of the classifier's ability to identify fraud instances while maintaining a low false positive rate. Consequently, classifiers with higher PR-AUC values are considered more effective in detecting fraud within imbalanced datasets.

The PR-AUC is calculated by summing up the areas of individual trapezoids formed by the Precision-Recall curve. Let's denote precision as P and recall as R. The PR-AUC can be mathematically expressed as:

$$PR\text{-}AUC = \sum_{i=1}^{n-1} \left(\frac{R_i + R_{i+1}}{2} \right) \cdot (P_{i+1} - P_i)$$

where n is the number of points on the Precision-Recall curve. The PR-AUC captures the overall performance of the classifier by considering the trade-off between precision and recall at different operating points.

b) F1 Score and F2 Score

F-beta score is a metric that combines precision and recall into a single value, allowing us to assess the performance of a classifier in a more balanced manner. It provides a flexible way to emphasize either precision or recall, depending on the specific requirements of the problem at hand. The F-beta score is calculated using the following formula:

$$F_\beta \ Score = \frac{(1 + \beta^2) \cdot Precision \cdot Recall}{(\beta^2 \cdot Precision) + Recall}$$

In this formula, beta represents the weight assigned to the recall component of the score. A higher beta value emphasizes recall more, while a lower beta value places more importance on precision. By adjusting the beta value, we can customize the score to suit the needs of our particular problem.

The F1 score and F2 score are specific cases of the F-beta score. The F1 score corresponds to the case when beta is set to 1, resulting in equal weight given to precision and recall. It is calculated as:

$$F1 \ Score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

The F2 score, on the other hand, is obtained by setting beta to 2, indicating a higher emphasis on recall. This score is useful in scenarios where correctly identifying positive cases is more important, even at the cost of more false positives. The formula for the F2 score is:

$$F2 \ Score = \frac{5 \cdot Precision \cdot Recall}{4 \cdot Precision + Recall}$$

By considering both precision and recall, the F1 Score and F2 Score provide comprehensive measures of a classifier's ability to identify fraudulent transactions accurately in imbalanced datasets.

C. Hyperparameters Tuning

In this section, the hyperparameters of various models, namely XGBoost, CatBoost, KNN, SVM, and Decision Tree, were tuned using a Bayesian search approach. The objective was to find the optimal combination of hyperparameters that maximized the performance of each model on the given dataset.

1) Bayesian Search

Bayesian search, also known as Bayesian optimization, is a technique used to efficiently search for the best set of hyperparameters for a machine learning model. Unlike grid search or random search, which exhaustively explore the entire hyperparameter space, Bayesian search uses a probabilistic model to estimate the performance of different hyperparameter configurations and selects the most promising ones for evaluation.

The main idea behind Bayesian search is to construct a surrogate model, typically a Gaussian process, that approximates the unknown function mapping hyperparameters to model performance. This surrogate model is updated iteratively as new evaluations of hyperparameter configurations are obtained. By leveraging the information from previous evaluations, Bayesian search focuses on promising regions of the hyperparameter space, leading to faster convergence towards the optimal solution.

The Bayesian search process involves two key components: an acquisition function and a surrogate model. The acquisition function determines the next hyperparameter configuration to evaluate based on the uncertainty and potential improvement in performance. Commonly used acquisition functions include Expected Improvement (EI), Probability of Improvement (PI), and Upper Confidence Bound (UCB). The surrogate model captures the probabilistic relationship between hyperparameters and model performance, allowing for efficient exploration and exploitation of the hyperparameter space.

Mathematically, Bayesian search aims to find the hyperparameter configuration $\mathbf{x}^*$ that maximizes the performance function $f(\mathbf{x})$, where $\mathbf{x}$ represents a set of hyperparameters. The acquisition function guides the search by balancing exploration and exploitation:

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} \text{Acquisition}(\mathbf{x}) \tag{11}$$

The process continues iteratively until a stopping criterion is met, such as a maximum number of iterations or a convergence threshold. At the end of the search, the hyperparameter configuration with the highest evaluated performance is selected as the best set of hyperparameters for the model.

2) XGBoost

XGBoost, a powerful gradient boosting algorithm, underwent a hyperparameter tuning process using Bayesian search to identify the optimal configuration. The key hyperparameters considered in the search were the maximum depth (max_depth), learning rate (lr), and number of estimators (n_estimators).

The Bayesian search algorithm efficiently explored the hyperparameter space, aiming to find the best combination that maximizes the model's predictive performance. For XGBoost, the following hyperparameters were used:

- Maximum Depth (max_depth): The range considered for max_depth during the search was from 1 to 10. After thorough evaluation, the best setting was determined to be a max_depth of 9. Setting the max_depth to 9 allows XGBoost to build trees with a depth of up to 9 levels, enabling the model to capture complex patterns and relationships in the data effectively.
- Learning Rate (lr): A fixed learning rate of 0.1 was utilized in the tuning process. The learning rate determines the step size at each boosting iteration and influences the model's convergence speed. A lower learning rate can potentially improve the model's accuracy, but it may require more boosting iterations to achieve the same performance.
- Number of Estimators (n_estimators): The number of estimators refers to the number of boosting iterations. In this case, a value of 200 was set for n_estimators. Increasing the number of estimators can enhance the model's predictive power, but it also increases the risk of overfitting. A suitable balance is crucial to achieve optimal performance.

3) CatBoost

CatBoost, a robust gradient boosting algorithm designed for categorical data, also underwent a hyperparameter tuning process using Bayesian search. The search aimed to identify the optimal configuration by considering the maximum depth (max_depth) hyperparameter.

The Bayesian search algorithm efficiently explored the hyperparameter space, seeking the best combination that maximizes CatBoost's predictive performance. The specific hyperparameter considered in the search was:

- Maximum Depth (max_depth): The range considered for max_depth during the search was from 1 to 10. After thorough evaluation, the best setting was determined to be a max_depth of 10. By setting the max_depth to 10, CatBoost can build trees with a depth of up to 10 levels, allowing the model to capture intricate relationships and patterns in the data more effectively.
- Learning Rate (lr): A fixed learning rate of 0.1 was utilized in the tuning process. The learning rate determines the step size at each boosting iteration and influences the model's convergence speed. A lower learning rate can potentially improve the model's accuracy, but it may require more boosting iterations to achieve the same performance.
- Number of Estimators (n_estimators): The number of estimators refers to the number of boosting iterations. In this case, a value of 200 was set for n_estimators. Increasing the number of estimators can enhance the model's predictive power, but it also increases the risk of overfitting. A suitable balance is crucial to achieve optimal performance.

Through the employment of Bayesian search, CatBoost's hyperparameters were fine-tuned, and the optimal setting of max_depth was determined to be 10. This enables CatBoost to achieve enhanced predictive performance by leveraging deeper trees to capture more complex relationships within the data.

4) SVM

Support Vector Machines (SVM) underwent a hyperparameter tuning process using Bayesian search to identify the optimal configuration. The key hyperparameters considered in the search were the regularization parameter (C).

The Bayesian search algorithm efficiently explored the hyperparameter space, aiming to find the best combination that maximizes SVM's predictive performance. For SVM, the following hyperparameters were tested:

- Regularization Parameter (C): The search considered values of C ranging from 0.1 to 1. After thorough evaluation, it was determined that a value of 1.0 for C yielded the best performance. The regularization parameter C controls the trade-off between achieving a low training error and a low testing error. A value of 1.0 indicates a balanced regularization, allowing SVM to generalize well on the given dataset.
- Kernel: The search also explored different kernel functions for SVM, including linear and radial basis function (RBF) kernels. The linear kernel is effective when the data is linearly separable in the input space, while the RBF kernel is more flexible and can handle nonlinear decision boundaries by mapping the data into a higher-dimensional feature space. The search evaluated the performance of SVM with both kernel options, and the RBF kernel was found to provide better results on the given dataset.

By leveraging Bayesian search, the hyperparameters of SVM were fine-tuned. The best configuration was identified as

C=1.0 with the RBF kernel. These optimized hyperparameters enable SVM to effectively learn and generalize from the given data, enhancing its predictive capabilities.

5) K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) underwent a hyperparameter tuning process using Bayesian search to identify the optimal configuration. The key hyperparameter considered in the search was the number of neighbors (n_neighbors).

The Bayesian search algorithm efficiently explored the hyperparameter space, aiming to find the best combination that maximizes KNN's predictive performance. For KNN, the following hyperparameter was tested:

- Number of Neighbors (n_neighbors): The search considered various values for the number of neighbors, ranging from 1 to 10. After thorough evaluation, it was determined that setting the number of neighbors to 3 yielded the best performance. This means that when making predictions, KNN considers the three closest neighbors based on their similarity to the target instance.

By leveraging Bayesian search and evaluating different values for the number of neighbors, the hyperparameters of KNN were fine-tuned. The best configuration was identified as having 3 neighbors. These optimized hyperparameters allow KNN to effectively analyze the local structure of the data and make accurate predictions based on the nearest neighbors.

6) Decision Tree

The Decision Tree model underwent a hyperparameter tuning process using Bayesian search to identify the optimal configuration. The key hyperparameter considered in the search was the maximum depth (max_depth).

The Bayesian search algorithm efficiently explored the hyperparameter space, aiming to find the best combination that maximizes the Decision Tree's predictive performance. For the Decision Tree model, the following hyperparameter was tested:

- Maximum Depth (max_depth): The search considered values for max_depth ranging from 1 to 10. After thorough evaluation, it was determined that a max_depth of 10 yielded the best performance. Setting the max_depth to 10 allows the Decision Tree to grow deeper, capturing more complex patterns in the data.

By leveraging Bayesian search and evaluating different values for the maximum depth, the hyperparameters of the Decision Tree model were fine-tuned. The best configuration was identified as having a max_depth of 10. These optimized hyperparameters enable the Decision Tree model to effectively capture intricate relationships in the data and make accurate predictions.

D. Comparison

The performance of various classifiers on the dataset is summarized in the table II. Despite the class imbalance, XGBoost and CatBoost consistently outperformed the other classifiers across all performance metrics. This indicates their robustness in handling imbalanced datasets and their ability to effectively identify the positive instances despite their scarcity.

TABLE II
PERFORMANCE OF DIFFERENT CLASSIFIERS ON THE DATASET

	XGBoost	**CatBoost**	**Decision Tree**	**KNN**	**SVM**
AUC	0.97	0.95	0.82	0.82	0.79
F1 score	0.95	0.94	0.92	0.89	0.82
F2 score	0.87	0.83	0.78	0.74	0.52

The highest AUC values achieved by XGBoost (0.97) and CatBoost (0.95) demonstrate their capability to differentiate between the positive and negative classes, even in the presence of class imbalance. This suggests that they are more adept at capturing the underlying patterns and subtle characteristics associated with fraud transactions, resulting in a better separation between the two classes.

Furthermore, when considering the F1 score, which provides a balanced measure of precision and recall, XGBoost (0.95) and CatBoost (0.94) again exhibited superior performance. This indicates their ability to achieve both high precision (minimizing false positives) and high recall (capturing a larger proportion of true positives) despite the class imbalance. The relatively higher F1 scores of these classifiers imply that they can effectively identify fraud transactions while keeping the number of misclassifications low.

Even in terms of the F2 score, which emphasizes recall and is particularly relevant when the minority class (fraud transactions) is of greater importance, XGBoost (0.87) and CatBoost (0.83) outperformed the other classifiers. This indicates their strength in correctly identifying a larger proportion of the positive instances, which is crucial in fraud detection scenarios.

Moreover, the superior performance of XGBoost and CatBoost on multiple evaluation metrics suggests that they excel in capturing complex relationships and distinguishing patterns within the imbalanced dataset. This is particularly valuable in fraud detection tasks where the positive instances (fraudulent transactions) are often scarce compared to the negative instances (legitimate transactions). Their robustness in handling class imbalance allows them to effectively identify the positive instances, minimizing the risk of false negatives and ensuring that fraudulent activities are detected.

The high AUC values obtained by XGBoost and CatBoost further highlight their ability to discriminate between the positive and negative classes. AUC measures the model's ability to rank instances correctly, and the high AUC scores indicate that these classifiers have a strong capability to differentiate between fraudulent and legitimate transactions.

Overall, the consistent outperformance of XGBoost and CatBoost across various performance metrics, such as AUC, F1 score, and F2 score, underscores their reliability and effectiveness in detecting fraud. These models not only exhibit high precision and recall but also demonstrate their capacity to handle imbalanced datasets and effectively identify the minority class. Their superior performance makes XGBoost and CatBoost promising choices for fraud detection tasks and highlights their robustness in addressing the challenges posed

by imbalanced datasets.

VI. Conclusion

In conclusion, XGBoost and CatBoost consistently demonstrated superior performance compared to other classifiers when considering the class imbalance present in the dataset. Their high AUC values, F1 scores, and F2 scores indicate their effectiveness in accurately detecting fraudulent transactions, even in the presence of a significantly larger number of non-fraud instances. These findings highlight the suitability of XGBoost and CatBoost for fraud detection tasks in imbalanced datasets, providing reliable and robust classification results. However, further analysis and evaluation, including techniques such as oversampling or cost-sensitive learning, may be necessary to address the class imbalance issue more effectively and ensure the generalizability of these findings to other imbalanced datasets.

VII. Acknowledgment

We would like to express our gratitude to Dr. Tiep Nguyen Vinh from the University of Information Technology for instructing and guiding the project implementation team. His expertise and valuable insights greatly contributed to the success of this project.

References

[1] Li, Y., Zhao, X., Guo, M. (2018). A hybrid model for credit card fraud detection. *Applied Soft Computing*, 67, 153-162.

[2] Malik, H., Shehzad, K., Aslam, N. (2019). A novel machine learning ensemble model for fraud detection in credit card transactions. *Journal of Computational Science*, 34, 45-56.

[3] Huang, Y., Li, J., Zhao, Y., Li, Y., Li, J. (2019). Fraud detection for online transactions using XGBoost. *Journal of Computational Science*, 35, 101-111.

[4] Jiang, Z., Cao, Z., Huang, Y., Liu, Y. (2020). Fraud detection for online transactions using CatBoost. *Information Sciences*, 508, 61-73.

[5] Akhtar, N., Idris, M. Y. I., Khan, M. K. (2016). Credit card fraud detection using K-nearest neighbor and majority voting. *Expert Systems with Applications*, 54, 196-207.

[6] Han, Y., Hong, J. (2020). Credit card fraud detection using support vector machine with undersampling and rotation forest. *Symmetry*, 12(2), 209.

[7] Bhattacharyya, S., Kalita, J. K. (2011). A fast and high-performance classifier for suspicious URL detection. *Expert Systems with Applications*, 38(5), 5491-5500.

[8] Phua, C., Alahakoon, D., Lee, V. (2004). Minority report in fraud detection: Classification of skewed data. *ACM SIGKDD Explorations Newsletter*, 6(1), 50-59.

[9] Durgesh K. Srivastava, Lekha "DATA CLASSIFICATION USING SUPPORT VECTOR MACHINE" Ass. Prof., Department of CSE/IT, BRCM CET, Bahal, Bhiwani, Haryana, India-127028