International Journal of Multidisciplinary Research and Growth Evaluation.

A novel credit card fraud detection using supervised machine learning model

Mohammed Siddiq Siraj [1*], Mohammed Wahaj haqqani [2], Dr. Khaja Mizbahuddin Quadry [3]
[1, 2] BECSE (AI & DS) CS & AI Department LIET OU Hyderabad, India
[3] Professor CSE Departments LIET OU Hyderabad, India

* Corresponding Author: **Mohammed Siddiq Siraj**

Article Info

ISSN (online): 2582-7138
Impact Factor: 5.307 (SJIF)
Volume: 05
Issue: 01
January-February 2024
Received: 05-11-2023;
Accepted: 09-12-2023
Page No: 313-324

Abstract
Financial fraud, especially in credit card transactions, is a growing concern. To tackle this, data mining techniques are used to automatically analyze large and complex financial datasets. Detecting credit card fraud is tricky because the patterns of normal and fraudulent behavior keep changing, and the data about fraud is much less common compared to legitimate transactions Several techniques were tried on a dataset from European cardholders, including Decision Tree, Random Forest, SVC, XGBoost, K-Nearest Neighbors, and Logistic Regress The dataset had information from 284,786 credit card transactions. To address the challenges, six advanced data mining approaches (Logistic Regression, K-Nearest Neighbors, Support Vector Classifiers, Decision Tree, Random Forests, and XGBoost) are evaluated. A comparative analysis is conducted to identify the best-performing model.

DOI: https://doi.org/10.54660/.IJMRGE.2024.5.1.313-324

Keywords: Financial fraud, machine, credit card, XGBoost

1. Introduction

1.1 Description

Financial fraud is a growing concern, especially with the increasing use of internet technology for credit card transactions. While online transactions have become more common, credit card fraud has also risen both online and offline. Various solutions and software have been developed to prevent fraud in industries like credit cards, retail, e-commerce, and insurance.

One effective method for tackling credit card fraud is data mining, which involves using mathematical algorithms to analyze available data and identify possible instances of fraud. The goal is to classify transactions into two categories: legitimate and fraudulent. Different techniques, such as Genetic Algorithm, Artificial Neural Network, and Frequent Item set Mining, and others, have been employed for credit card fraud detection.

However, credit card transaction datasets are often limited, imbalanced, and skewed. Selecting the right features and metrics for evaluating model performance is crucial. Challenges in credit card fraud detection include the dynamic nature of fraudulent behavior, where fraudulent transactions can resemble legitimate ones. The performance of detection techniques is also influenced by the sampling approach, variable selection, and the chosen detection technique.

In the conducted experiments, Logistic Regression showed an accuracy of 94.4%, SVC had 93.4%, KNN achieved 93.9%, Decision Tree resulted in 91.9%, and Random Forest had 92.9%. XGBoost emerged as the most accurate with 94.95%. However, when evaluating the learning curves, it was observed that XGBoost, Random Forest, and Decision Tree tended to over fit, while KNN exhibited better learning. Consequently, it was concluded that KNN is the most suitable model for the system.

1.2 Problem formulation

Our project is about figuring out if the ways we currently catch credit card fraud are really good or if we can do better using Machine Learning. We want to use smart computers to predict fraud because we think it might work better than what we're doing now. This matters for anyone who cares about keeping credit card transactions safe.

1.3 Proposed Solution

In this paper, the authors propose a method for detecting fraudulent activities in credit card transactions. They compare various machine learning algorithms, including Logistic Regression, Decision Trees, and Random Forest, to find the most effective one for credit card merchants to identify fraud.

Here's a simplified explanation of the algorithm steps:

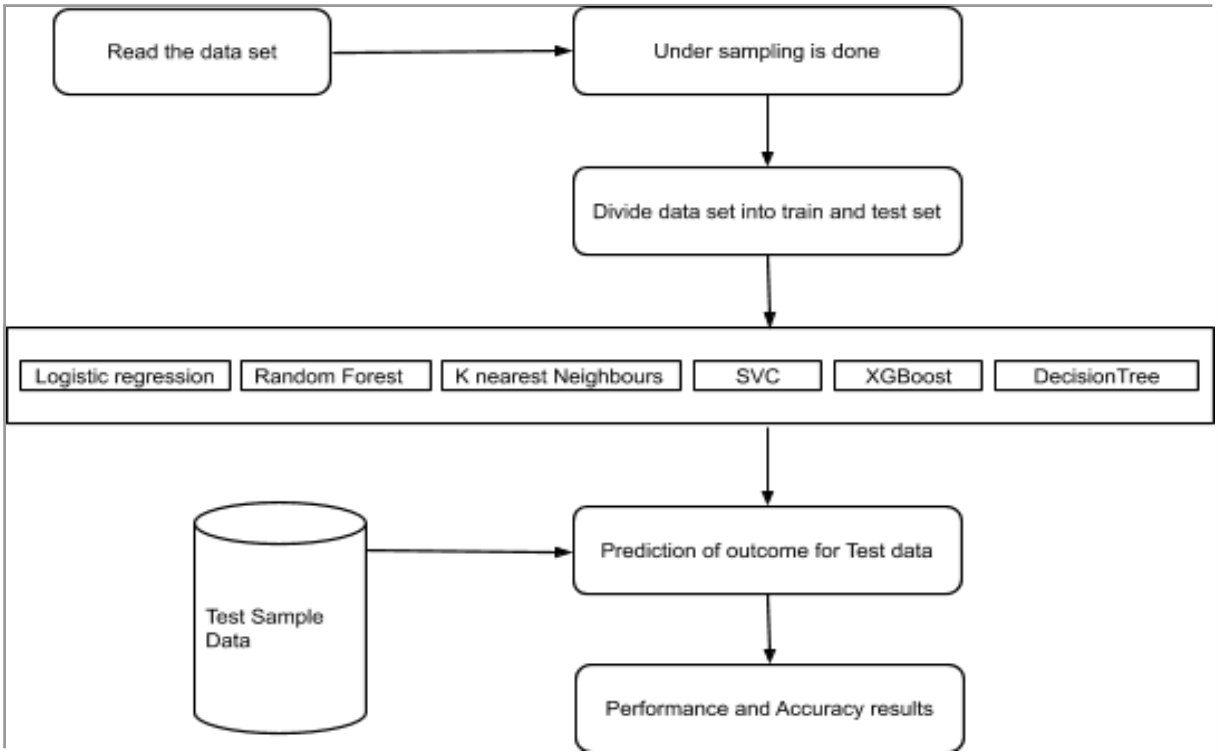


Fig 1 System Architecture

Random Sampling: To create a balanced dataset, randomly select a subset of the data.

Split Dataset: Divide the data into two parts - one for training the algorithm and the other for testing its performance.

Evaluate Accuracy: Calculate accuracy and other performance metrics to assess how well the algorithms work with the given data.

Identify Best Algorithm: Determine the most efficient algorithm based on its performance on the dataset.

The goal is to provide credit card merchants with a reliable algorithm that can effectively detect and prevent fraudulent transactions, ensuring the security of credit card systems. The paper also includes a diagram illustrating the overall system framework.

1.4. Scope of the project

Credit card fraud detection is a challenging problem due to limited data and deceptive tactics by fraudsters. The dataset is small, making it difficult to identify patterns. Additionally, fraudulent entries may resemble legitimate behavior, complicating pattern recognition. The issue is exacerbated by limited public access to datasets and restricted sharing of research results, hindering model benchmarking.

Security concerns impede the exchange of ideas in fraud detection, making method improvement challenging. Furthermore, the evolving nature of datasets constantly alters normal and fraudulent behavior profiles. For instance, a legitimate transaction in the past may now appear fraudulent, and vice versa.

To address these challenges, six advanced data mining approaches (Logistic Regression, K-Nearest Neighbors, Support Vector Classifiers, Decision Tree, Random Forests, and XGBoost) are evaluated. A comparative analysis is conducted to identify the best-performing model.

2. Literature Survey

Research on credit card fraud detection has focused on various techniques, particularly neural networks, data mining, and distributed data mining. Machine learning methods, such as artificial neural networks, rule-induction techniques, decision trees, logistic regression, and support vector machines, are commonly used for this purpose.

In one study by Behrouz, Emad, and Sahil, supervised machine learning algorithms were employed for credit card fraud detection. They created a super classifier using ensemble learning methods and compared its performance with various algorithms found in literature.

Navanshu Khare and Saad Yunus Sait explained the mathematics and functioning of the algorithms used in their paper on credit card fraud detection.

Another study by Siddhartha Bhattacharyya and team compared Support Vector Machine, Random Forest, and Logistic Regression. They found that Random Forest showed the highest accuracy, followed by Logistic Regression and Support Vector Machine.

Y. Sahin and E. Duman proposed fraud detection using a combination of Support Vector Machines and Decision Trees. They observed that Decision Trees outperformed SVMs when the dataset was small, but as the dataset size increased, SVM achieved higher accuracy than decision trees.

3. System Design

3.1 Functional Requirements

Accurate Predictions: The model must provide precise and reliable predictions.
Graphical Visualization: The application should display predicted results and general data through graphical visuals.
User Input: Users must have the ability to input values for predictions.

3.2 Non-Functional Requirements

Availability: The system should be accessible to any transaction verification system.
Correctness: The system's accuracy should be maximized for better predictions.
Maintainability: The system must maintain an accurate history of records.
Usability: The system should meet the needs of a wide range of banking systems.

3.3 Specific Requirements: Software Requirements:
Languages: Python-3
Operating Systems: Windows, Linux, etc.
Back End Software: Anaconda, Jupiter notebook.
Hardware Requirements:
CPU: Intel Pentium IV 600MHz
Hard Disk Space: 20 GB or more
Memory: 4 GB RAM

3.4. Block Diagrams

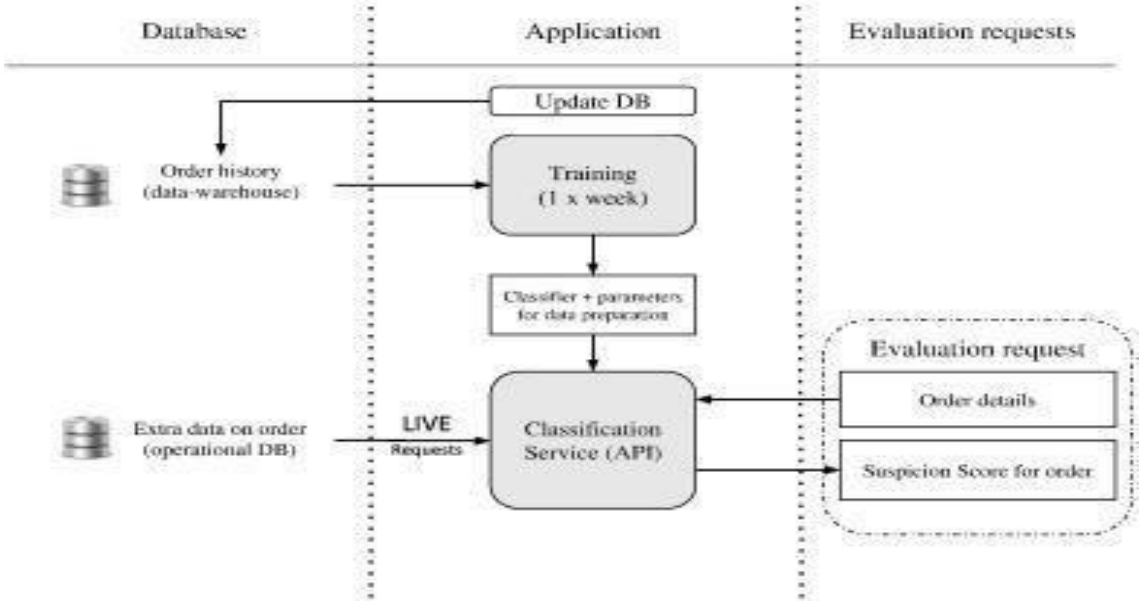


Fig 2: Activity Diagram

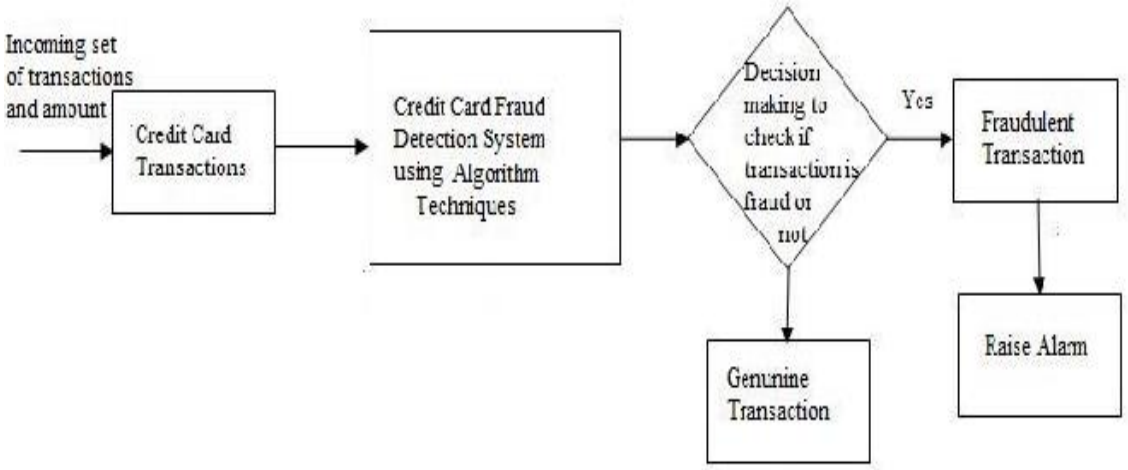


Fig 3: Flow Diagram

Activity Diagram

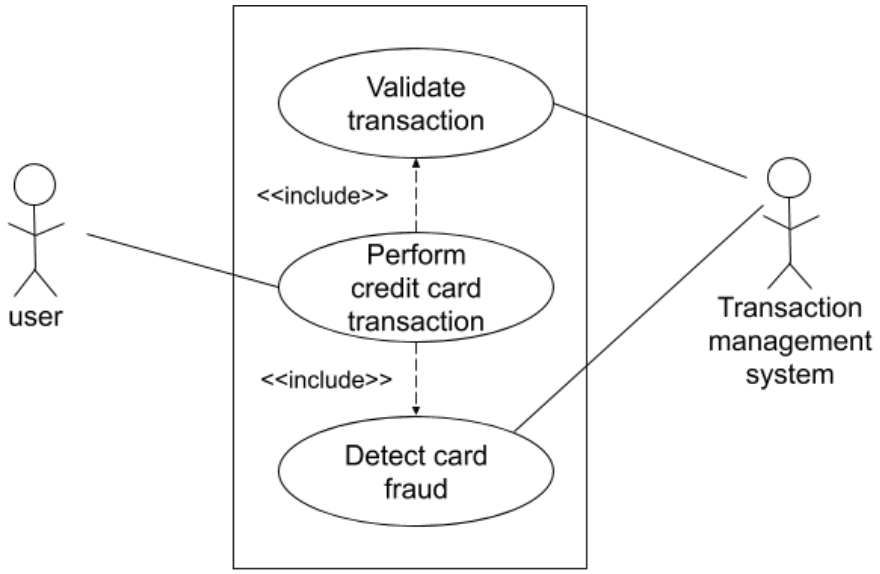


Fig 4: Use case diagram for fraud detection

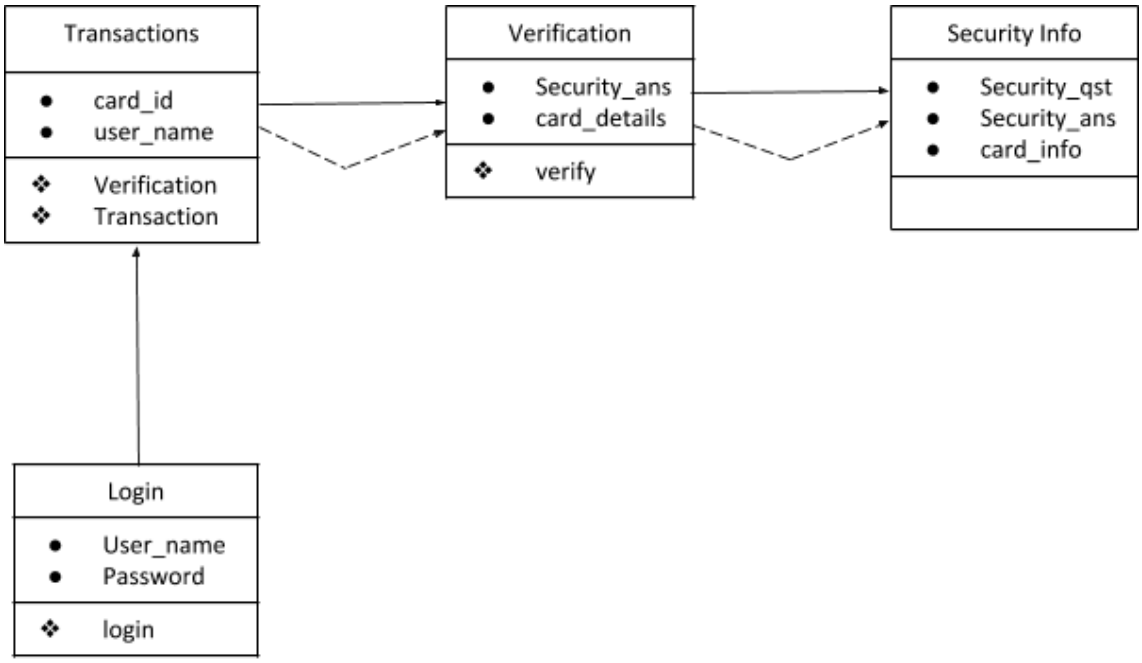


Fig 5: Class Diagram

A gram (Figure 2): (Provide a simple description of the activity diagram, or if possible, insert a diagram here.)
Flow Diagram (Figure 3): (Describe the key steps in the flow diagram or insert the diagram here.)
These diagrams help visualize the flow of activities and processes in the system, making it easier to understand and implement.

4. Implementations Details

4.1 Algorithms

Algorithms: Machine learning is the ability of a system to learn and improve from experience without explicit programming. It involves creating computer programs that access and learn from data on their own. A classifier, in this context, is an algorithm that categorizes input data, acting as a mathematical function to map input into categories. It's a form of supervised learning, utilizing a training set with correctly identified observations.

Logistic Regression

Logistic Regression: Logistic Regression is a supervised classification method that predicts the probability of a binary outcome (like yes/no or true/false) based on input data. Unlike linear regression, which uses a straight line, logistic regression shows a curve. It analyzes the relationship between multiple independent variables and produces logistic curves that plot values between zero and one. There are different types of logistic regression models, such as binary logistic, multiple logistic and binomial logistic, each serving various purposes in estimating probabilities.

Definition: A supervised classification method predicting the

probability of a binary outcome based on input variables.
Key Feature: Utilizes logistic curves to represent the probability of an outcome.

Support Vector Machine (SVM)

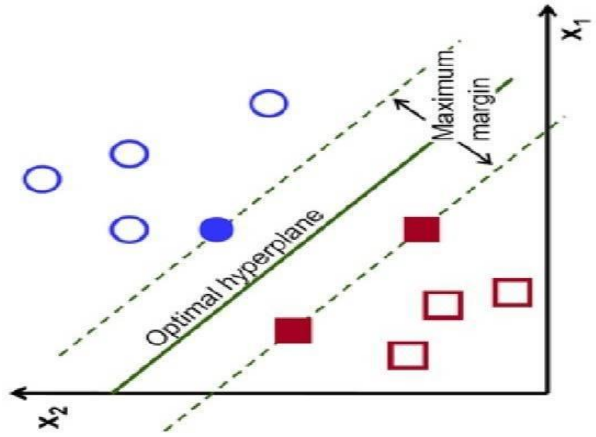


Fig 6

Definition: A supervised learning algorithm for classification and regression, creating a hyper plane to separate different classes.
Key Feature: Represents training data points in space, mapping them to ensure a gap between classes.

Decision Tree
Definition: An algorithm using a tree-like model to predict outcomes by following decision rules.

$$Entropy(S) = \sum_{i=1} -p_i \log_2 p_i$$

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Key Feature: Represents decisions and possible outcomes in a tree structure, facilitating interpretation.

Random Forest
Definition: An ensemble algorithm consisting of decision tree classifiers, correcting over fitting.
Key Feature: Builds multiple trees on random subsets of data, combining their predictions for more robust results.

K-Nearest Neighbor Classifier (KNN)
Definition: An instance-based learning algorithm classifying based on similarity measures.
Key Feature: Uses Euclidean distance to find k-nearest neighbors and predicts the majority class.

XG Boost
Definition: Short for Extreme Gradient Boosting, a boosting algorithm that sequentially improves predictions.
Key Feature: Adds new trees to complement existing ones, handling missing values for numeric data.

4.2 Working of the Project

- The project aims to detect fraudulent credit card transactions. The dataset, containing transactions by European cardholders, is highly imbalanced, with only 0.172% being frauds. To address this, the project applies various algorithms after scaling and distributing the data.
- Handling Class Imbalance in Credit Card Fraud Detection:
  Challenges of Imbalanced Dataset: Our original dataset is highly imbalanced, with non-fraudulent transactions making up 99.83% and fraudulent ones only 0.17%. If we use this imbalance directly for predictive models, algorithms might not effectively identify fraud because they'll assume most transactions are not fraudulent. This situation can lead to overfitting, where the model becomes too biased towards non-fraudulent cases.
- Accuracy Paradox: Simply applying classification algorithms directly to such imbalanced data can result in what's known as the accuracy paradox. High accuracy in predicting non-fraudulent transactions may seem good, but it's misleading. For instance, predicting all cases as non-fraud would yield 99% accuracy, but it's not useful. Precision and recall become more meaningful in such cases.
- Class Imbalance and Bias: The underlying problem is the class imbalance between non-fraud (negative class) and fraud (positive class) cases. Accuracy, which is the ratio of correctly predicted instances to the total instances, can be biased when there's a significant class imbalance. Precision (accuracy of positive predictions) and recall (sensitivity or true positive rate) provide better insights.
  Distribution of Features:
- Examining the distribution of transaction amounts and time gives us an understanding of how skewed these features are. Skewed features can impact the model's ability to identify patterns effectively. Techniques to address skewed distributions will be implemented in future reports. Dealing with Imbalance:
- To address the class imbalance problem, one effective strategy is class distribution, where the minority class (fraudulent transactions) is oversampled. By increasing the number of minority class examples, we enhance the algorithm's chances of correctly predicting fraud.
- In summary, the challenge lies in the imbalance between fraudulent and non-fraudulent transactions. Directly applying models to such imbalanced data can lead to misleading accuracy. Precision, recall, and addressing skewed features are crucial. Future efforts will focus on implementing techniques to handle skewed distributions and maintain a balanced class distribution for effective credit card fraud detection.

```
...: dataset.describe()
Out[1]:
                Time             V1  ...          Amount          Class
count  284807.000000   2.848070e+05  ...  284807.000000  284807.000000
mean    94813.859575   3.919560e-15  ...      88.349619       0.001727
std     47488.145955   1.958696e+00  ...     250.120109       0.041527
min         0.000000  -5.640751e+01  ...       0.000000       0.000000
25%     54201.500000  -9.203734e-01  ...       5.600000       0.000000
50%     84692.000000   1.810880e-02  ...      22.000000       0.000000
75%    139320.500000   1.315642e+00  ...      77.165000       0.000000
max    172792.000000   2.454930e+00  ...   25691.160000       1.000000
```

```
In [2]: dataset.head()
Out[2]:
   Time        V1        V2        V3  ...         V27       V28  Amount  Class
0   0.0 -1.359807 -0.072781  2.536347  ...    0.133558 -0.021053  149.62      0
1   0.0  1.191857  0.266151  0.166480  ...   -0.008983  0.014724    2.69      0
2   1.0 -1.358354 -1.340163  1.773209  ...   -0.055353 -0.059752  378.66      0
3   1.0 -0.966272 -0.185226  1.792993  ...    0.062723  0.061458  123.50      0
4   2.0 -1.158233  0.877737  1.548718  ...    0.219422  0.215153   69.99      0
```

Given the class imbalance ratio, we recommend measuring the accuracy using the Area Under the Precision-Recall Curve (AUPRC). Confusion matrix accuracy is not meaningful for unbalanced.

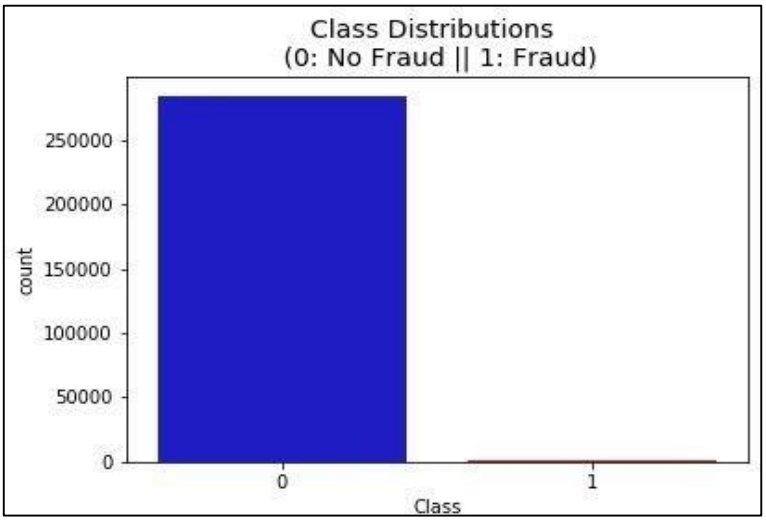


Fig 7

No frauds 99.83% of dataset Frauds 0.17% of dataset
Our dataset is heavily imbalanced, with 99.83% non-fraudulent transactions and only 0.17% fraudulent ones. Using this data directly for predictions might lead to errors and over fitting, where the model assumes most transactions are not fraud. In credit card fraud detection, this imbalance can result in a high accuracy score that doesn't effectively catch fraud. We need to address this imbalance to ensure our model detects patterns indicating fraud accurately.

```
KNearest
[[56864     0]
 [   93     5]]
accuracy_score: 0.9983673326077034
miscalssification: 0.0016326673922966162
Sensitivity :  1.0
Specificity :  0.05102040816326531
precision_score: 1.0
f1_score: 0.09708737864077671

LogisiticRegression
[[56853    11]
 [   46    52]]
accuracy_score: 0.9989993328885924
miscalssification: 0.0010006671114075605
Sensitivity :  0.9998065559932471
Specificity :  0.5306122448979592
precision_score: 0.8253968253968254
f1_score: 0.6459627329192548
```

```
XGBClassifier
[[56858     6]
 [   18    80]]
accuracy_score: 0.9995786664794073
miscalssification: 0.0004213335205927038
Sensitivity :  0.9998944850872257
Specificity :  0.8163265306122449
precision_score: 0.9302325581395349
f1_score: 0.8695652173913043

RandomForestClassifier
[[56862     2]
 [   21    77]]
accuracy_score: 0.9995962220427653
miscalssification: 0.0004037779572346744
Sensitivity :  0.9999648283624085
Specificity :  0.7857142857142857
precision_score: 0.9746835443037974
f1_score: 0.8700564971751412
```

The accuracy paradox highlights a problem where high accuracy in a model may not guarantee its effectiveness. This occurs when a simple model, even with high accuracy, is too basic to be useful. For instance, if category A dominates with a 99% incidence, predicting every case as category A will yield 99% accuracy but lacks meaningful insights. Precision and recall are better metrics in such cases. The issue arises due to class imbalance, where one class is much more prevalent. Techniques to address skewed features will be explored in future reports.

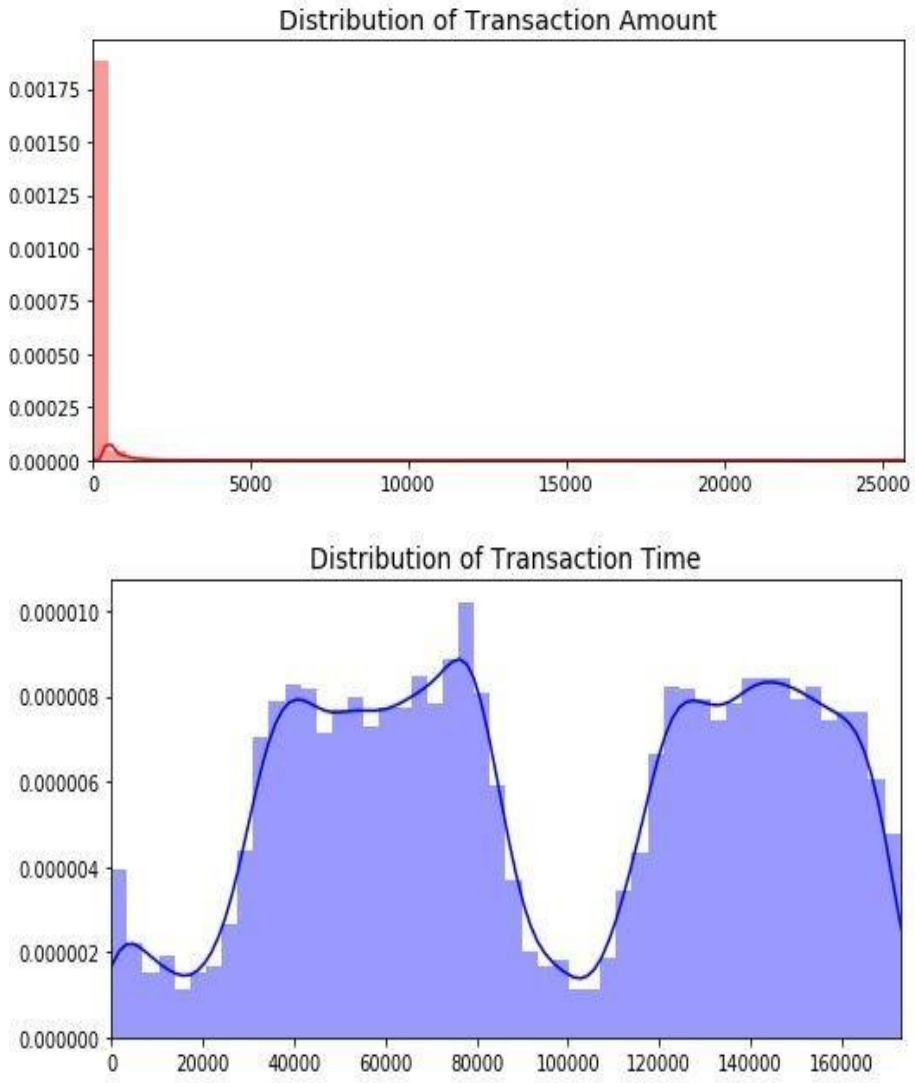


Fig 8: Distribution of Transaction Amount and Time

Scaling and Distributing

In this step, we're preparing our data for analysis. Since the original dataset is heavily imbalanced with mostly non-fraudulent transactions, we create a subsample with an equal number of fraud and non-fraud cases (a 50/50 ratio). This helps prevent issues like overfitting, where the model assumes there are no frauds, and incorrect correlations due to the imbalance.

To ensure fair scaling, we use RobustScaler, which handles outliers well by removing the median and scaling based on the interquartile range. QuantileTransformer is also applied, robust to outliers, and automatically adjusts any outliers.

Summary

The dataset is then split, and a new subsample is created by randomly selecting 492 cases each of fraud and non-fraud. This balanced subsample is used for further analysis**.**

```
from sklearn.preprocessing import StandardScaler, RobustScaler
rob_scaler = RobustScaler()
dataset['scaled_amount'] = rob_scaler.fit_transform(dataset['Amount'].values.reshape(-1,1))
dataset['scaled_time'] = rob_scaler.fit_transform(dataset['Time'].values.reshape(-1,1))
dataset.drop(['Time','Amount'], axis=1, inplace=True)
scaled_amount = dataset['scaled_amount']
scaled_time = dataset['scaled_time']
dataset.drop(['scaled_amount', 'scaled_time'], axis=1, inplace=True)
dataset.insert(0, 'scaled_amount', scaled_amount)
dataset.insert(1, 'scaled_time', scaled_time)
# Amount and Time are Scaled!
```

```
In [6]: dataset.head()
Out[6]:
        scaled_amount  scaled_time         V1  ...         V27       V28  Class
42948       -0.167680    -0.509675  -0.256652  ...   -0.547145 -0.380498      0
74625       -0.293579    -0.340923  -0.357039  ...    0.243853  0.085171      0
236178      -0.296793     0.751959  -0.443426  ...   -0.665420 -0.268262      0
192669       0.041920     0.529940   2.334822  ...    0.006936 -0.071813      0
15736        1.089779    -0.675866 -23.914101  ...    1.458076  0.430315      1
```

Before using the Random under sampling technique, it's crucial to set aside the original data for testing. Even though we split the data during the under sampling or oversampling processes, our objective is to train models on the adjusted datasets and evaluate them on the original test set. A common and effective strategy for handling imbalanced data is Random under Sampling, which involves reducing the instances of the majority class. Oversampling the minority class before cross-validation can lead to over fitting issues.

```
In [8]: X = dataset.drop('Class', axis=1)
   ...: y = dataset[['Class']]
   ...: from sklearn.model_selection import StratifiedShuffleSplit
   ...: sss = StratifiedShuffleSplit(n_splits=5, test_size=0.2, random_state=42)
   ...: for train_index, test_index in sss.split(X, y):
   ...:     print("Train:", train_index, "Test:", test_index)
   ...:     original_Xtrain, original_Xtest = X.iloc[train_index], X.iloc[test_index]
   ...:     original_ytrain, original_ytest = y.iloc[train_index], y.iloc[test_index]
   ...:
   ...: original_Xtrain = original_Xtrain.values
   ...: original_Xtest = original_Xtest.values
   ...: original_ytrain = original_ytrain.values
   ...: original_ytest = original_ytest.values
   ...: train_unique_label, train_counts_label = np.unique(original_ytrain,
return_counts=True)
   ...: test_unique_label, test_counts_label = np.unique(original_ytest,
return_counts=True)
   ...: print('-' * 100)
   ...: print('Label Distributions: \n')
   ...: print(train_counts_label/ len(original_ytrain))
   ...: print(test_counts_label/ len(original_ytest))

Train: [265518 180305  42664 ...  29062  13766  17677] Test: [263020  11378 147283 ... 274532 269819  64170]
Train: [ 72227 114282  16818 ... 264471 191914 284017] Test: [202638  32978 128121 ... 244024 127667  48318]
Train: [ 20895 114622 167683 ... 244502 178972 218506] Test: [284352  82483  90981 ... 171224 168807 271602]
Train: [122248 181660 194400 ... 104631 277586  29432] Test: [225673  63348  68025 ... 279451  77554  76043]
Train: [241684 223467 136928 ...  86495 160550  49633] Test: [157557 204860  83760 ... 251478 178967 216850]
----------------------------------------------------------------------------------------------------
Label Distributions:

[0.99827075 0.00172925]
[0.99827955 0.00172045]
```

Under Sampling

In this project phase, we're applying "Under Sampling" to balance our dataset and prevent over fitting.

First, we assess the class imbalance by checking the counts of each label using the "value_counts()" on the class column. Once we identify the number of fraud instances (Fraud = "1"), we aim for a 50/50 ratio. Therefore, we'll equalize non-fraud transactions to match the number of fraud transactions, resulting in 492 cases for each.

After implementing this technique, our data frame becomes a sub-sample with a balanced 50/50 ratio for the classes.

As a next step, we shuffle the data to evaluate if our models consistently maintain a certain accuracy across runs.

Note: "Random Under-Sampling" has a drawback – there's a risk that our classification models might not perform as accurately due to significant information loss (bringing 492 non-fraud transactions from 284,315 non-fraud transactions).

```
In [9]: dataset = dataset.sample(frac=1)
   ...: # amount of fraud classes 492 rows.
   ...: fraud_df = dataset.loc[dataset['Class'] == 1]
   ...: non_fraud_df = dataset.loc[dataset['Class'] == 0][:497]
   ...: normal_distributed_df = pd.concat([fraud_df, non_fraud_df])
   ...: # Shuffle dataframe rows
   ...: new_df = normal_distributed_df.sample(frac=1, random_state=42)
   ...: new_df.head()
   ...: new_df.describe()
   ...: print('Distribution of the Classes in the subsample dataset')
   ...: print(new_df['Class'].value_counts()/len(new_df))
Distribution of the Classes in the subsample dataset
0    0.502528
1    0.497472
Name: Class, dtype: float64
```

Now that we have our data frame correctly balanced, we can go further with our analysis and Data preprocessing.

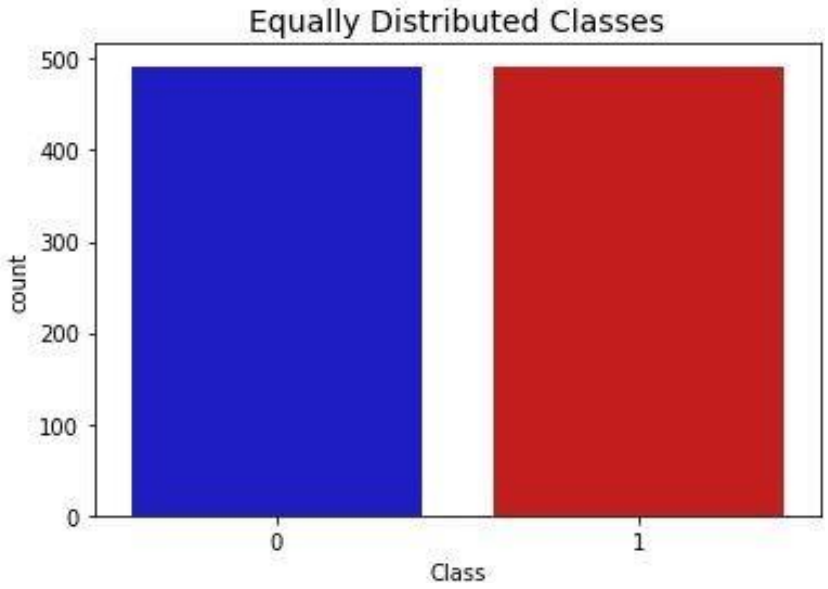


Fig 9

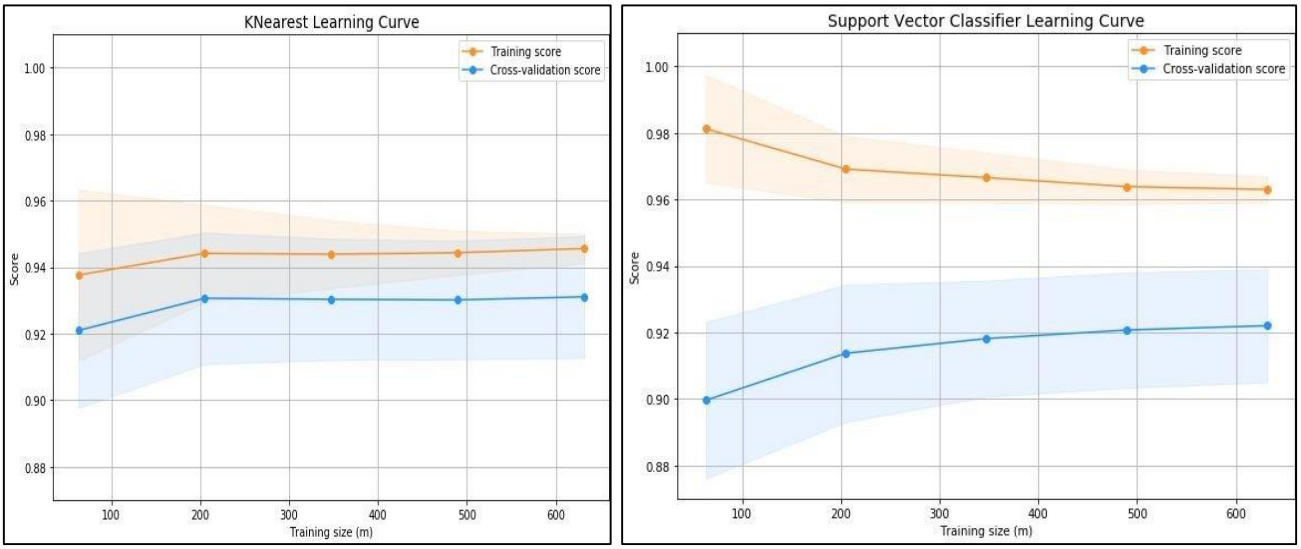
Now that we have sampled our data set, we apply our supervised learning models

```
In [26]: classifiers = {
   ...:     "RandomForestClassifier": RandomForestClassifier(n_estimators=300),
   ...:     "LogisiticRegression": LogisticRegression(),
   ...:     "KNearest": KNeighborsClassifier(),
   ...:     "Support Vector Classifier": SVC(),
   ...:     "DecisionTreeClassifier": DecisionTreeClassifier(),
   ...:     "XGBClassifier":XGBClassifier()
   ...: }
```

```
In [30]: for key, classifier in classifiers.items():
    ...:     classifier.fit(X_train, y_train)
    ...:     y_pred = classifier.predict(X_test)
    ...:     #metrics
    ...:     print("Classifiers: ", classifier.__class__.__name__)
    ...:     cm=confusion_matrix(y_test, y_pred)
    ...:     print(cm)
Classifiers:  DecisionTreeClassifier
[[84  7]
 [11 96]]
Classifiers:  RandomForestClassifier
[[90  1]
 [14 93]]
Classifiers:  LogisticRegression
[[88  3]
 [13 94]]
Classifiers:  KNeighborsClassifier
[[90  1]
 [16 91]]
Classifiers:  SVC
[[90  1]
 [14 93]]
Classifiers:  XGBClassifier
[[86  5]
 [12 95]]
```

Sampling techniques, scaling, and using various algorithms contribute to achieving accurate predictions in credit card fraud detection.

5. Result Analysis



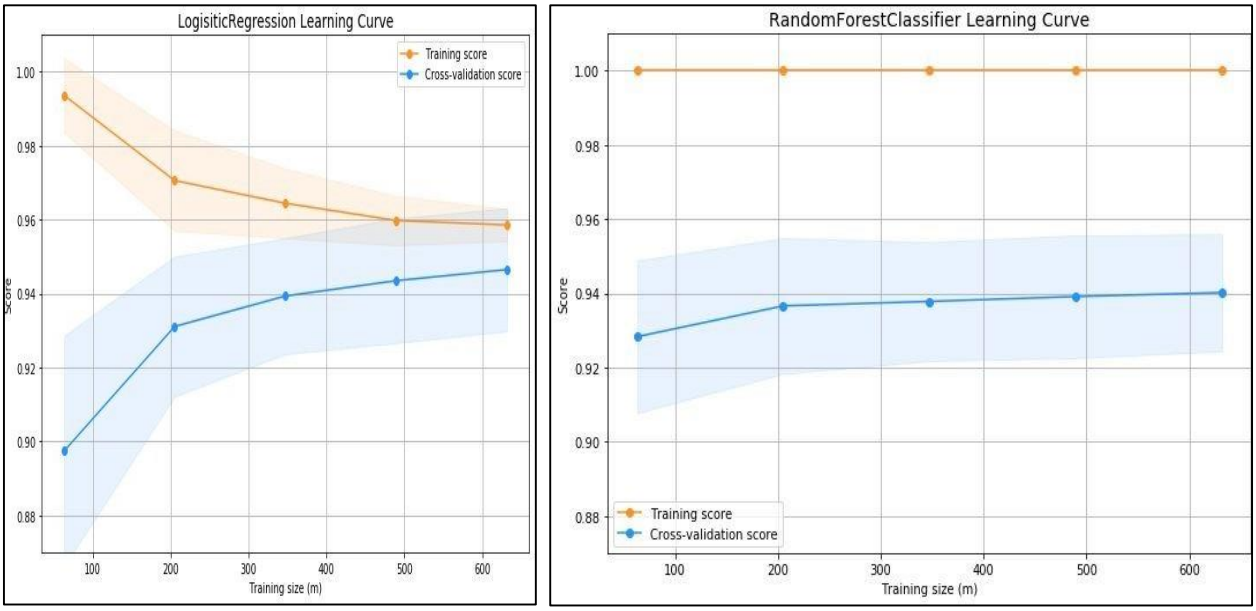


Fig 10: Learning Curves of Models

Learning Curves

Learning curves help understand how well the model performs on training and cross-validation sets.

A wider gap between the two scores may suggest over fitting (high variance), while low scores in both sets indicate under fitting (high bias).

The K Nearest Neighbor Classifier shows the best scores in both training and cross-validation sets.

Metrics

Metrics involve measuring true positive, true negative, false positive, and false negative to assess system or algorithm performance.

True Positive (TP) represents correctly identified fraudulent transactions.

True Negative (TN) denotes correctly identified legitimate transactions.

False Positive (FP) signifies legitimate transactions wrongly classified as fraudulent.

False Negative (FN) indicates fraudulent transactions wrongly classified as legitimate.

Table 1: Basic Metrics for Sampled Data Distribution

Metrics	Random Forest	Decision Tree	XGBoost	KNearest	SVC	Logistic Regression
True Positive	90	86	90	91	87	90
False Positive	1	5	1	0	4	1
False Negative	13	11	9	12	9	10
True Negative	94	96	98	95	98	97

Table 2: Accuracy Results for Sampled Data Distribution

Metrics	Random Forest	Decision Tree	XGBoost	KNearest	SVC	Logistic Regression
Accuracy	0.9293	0.9192	0.9495	0.9394	0.9343	0.9444
Sensitivity	0.8785	0.8972	0.9159	0.8878	0.9159	0.9065
Specificity	0.9890	0.9341	0.98901	1.0	0.9560	0.9890
Precision	0.9895	0.9505	0.9899	1.0	0.9608	0.9898
F1 Score	0.93069	0.9231	0.9515	0.9406	0.9378	0.9463

6. Future Scope

As the number of bank fraud and cybercrime cases continues to rise, the demand for a secure testing system is increasing. The proposed solution can be extended to provide a dual verification system, not only for customers (debit-ant) but also for sellers (credit-ant). It has the potential to become a routine security measure similar to OTP (One-Time Password).

Potential Enhancements and Applications

Duplex Verification: Extend the system to verify both customers and sellers in transactions, adding an extra layer of security.

Regular Usage: Implement the system as a standard security measure in day-to-day transactions, enhancing overall security for users.

Transaction History Analysis: Enable the system to assess the past transaction database, identifying and flagging potentially fraudulent transactions. It could serve as a valuable tool for evidence in fraud cases.

Optimization and Model Testing

Algorithm Optimization: Explore optimization techniques to improve the efficiency and accuracy of the proposed models.

Testing New Models: Continuously evaluate and test new machine learning models to stay updated with evolving fraud detection methodologies.

Simplified User Interface: Design a user-friendly interface

for easy adoption by customers and businesses, ensuring widespread usability.

Integration with Banking Systems: Collaborate with financial institutions to integrate the system seamlessly into existing banking systems, making it accessible to a broader user base.

7. Conclusion

Challenges in Fraud Detection

Despite various fraud detection techniques available today, none can completely identify frauds as they happen. Typically, fraud detection systems discover fraudulent activity after the fact. This is mainly due to the extremely small number of actual fraudulent transactions compared to the total number of transactions. Therefore, there's a need for technology that can detect fraudulent transactions in real-time to prevent them immediately, and at a minimal cost. The challenge extends to detecting various types of fraud, including online activities like phishing and site cloning, as well as detecting tampering with physical credit cards.

Drawbacks of Current Techniques

Most existing fraud detection techniques are not foolproof and may yield varying results in different environments. Their effectiveness heavily depends on the type of dataset used. For instance, in an under sampled dataset (where non-fraud transactions are deliberately reduced), models may struggle to accurately classify non-fraud transactions, potentially leading to false positives. This situation could result in regular customers having their cards blocked, causing dissatisfaction and increasing customer complaints.

Improving Accuracy

To address these issues, the next step involves performing outlier removal on the under sampled dataset to enhance accuracy in detecting non-fraud transactions.

Machine Learning Techniques Used

The paper utilizes various machine learning techniques, including Logistic Regression, Decision Tree, Random Forest, K-Nearest Neighbors (KNN), Support Vector Classifiers (SVC), and XGBoost, to detect credit card fraud. The evaluation is based on sensitivity, specificity, accuracy, and error rate.

Results and Conclusions

Logistic Regression achieves an accuracy of 94.4%, SVC at 93.4%, KNN at 93.9%, Decision Tree at 91.9%, and Random Forest at 92.9%.

XGBoost stands out with the highest accuracy of 94.95%.

However, learning curves analysis reveals that XGBoost, Random Forest, and Decision Tree tend to overfit, while others under fit.

Based on the overall performance, KNN is identified as the best model for the system.

Future Directions

The paper highlights the ongoing challenges in fraud detection and emphasizes the need for real-time detection technology. Further improvements, such as outlier removal, aim to enhance the accuracy of fraud detection systems. The study concludes by endorsing KNN as the most effective model, acknowledging the continuous evolution needed in fraud detection techniques.

In summary, the future scope involves expanding the system's capabilities, optimizing algorithms, testing new models, and ensuring user-friendly implementation to address the growing challenges of bank fraud and cybercrime.

8. References

1. Y Sahin, S Bulkan, E Duman. A cost-sensitive decision tree approach for fraud detection, Expert Syst. Appl. 2013; 40(15):5916-5923.
2. Sahil Dhankhad, Emad A. Mohammed and Behrouz Far, Supervised Machine Learning Algorithms for Credit Card Fraudulent Transaction Detection: A Comparative Study.
3. Navanshu Khare, Saad Yunus Sait, Credit Card Fraud Detection Using Machine Learning Models and Collating Machine Learning Models
4. Dal Pozzolo, Andrea, *et al*. Calibrating Probability with Undersampling for Unbalanced Classification. Computational Intelligence, 2015 IEEE Symposium Series on. IEEE, 2015.
5. Lakshmi SVS S1,Selvani Deepthi Kavila2 Machine Learning For Credit Card Fraud Detection System
6. SJKTJCW. Siddhatha Bhattacharya, Data Mining for credit card fraud: A comparative study, Elsevire. 2011; 50(3):602-613.
7. EDY Sahin. Detecting credit card fraud by decision trees, in Proceedings of the international multiconference of engineers and computer science, Hong Kong, 2011.