Banking Transaction Fraud Detection System Using Machine Learning

05/10/2025

John Anokye

1.0. Introduction

In recent years we have seen significant adoption of online transactions as it has revolutionized the banking industry. This adoption has however led to an increase in banking fraud. According to PwC's Global Economic Crime and Fraud Survey 2020, respondents reported losses ranging in the $ billions over the past years due banking fraud as businesses continue contactless payments. This paper introduces a secure, scalable, and production-ready machine learning system to detect fraudulent banking transactions in near real-time. This system was developed using Python, Scikit-learn, SQLite, and implemented industry-standard security practice. The system integrates data injection and preprocessing, machine learning model training, tuning, and evaluation, and secure handing of sensitive data.

The goal of this project is to automate online transaction fraud detection using machine learning models with high recall and interpretability. High recall means lower miss detection. Several models that will be evaluated include KNN, logistic regression as well as a deep learning model. The project will consist of two phases: a Machine Learning Modeling phase and an Application Development phase. The former evaluates several classification algorithms, while the latter implements the final system into a deployable, testable, and modular application.

2.0. Background and Problem Statement

In recent years, machine learning and artificial intelligence has virtually been used in every industry. From the industry of self-driving vehicles, space exploration, natural language processing, etc. There are many ways machine learning has been put to work; one way is fraud detection where it is used to detect changes in pattern. The dependency on e-commerce and online payments has increasingly grown and fraudulent behavior has become difficult to analyze and detect due to the dynamic nature of fraud.

This paper aims to provide a comparison among several machine learning models and its application in fraud detection. Validation technique's will be employed to provide the best performing model and best model parameters. Many existing systems suffer from poor recall and high false-negative rates, meaning they often fail to detect actual fraud. In that respect, the best performing model will be selected based on the one with a higher recall.

The dataset used for this study was obtained from Deutsche Bank but has several features anonymized. The dataset contains over 6 million records but have about 0.1%of the records labelled as fraudulent transactions. This severe class imbalance makes fraud detection particularly challenging. Another challenged that was identified relates to computing power as there was unsuccessful attempt in building a training model on 6million + records. I random sampling method was used to randomly sample a representative 600,000 records for the model training. The data also contains personally identifiable information (PII) and hashing was used to mask these PII fields.

The primary objective was to build a model with a low false-negative rate—i.e., minimizing undetected fraudulent transactions—while maintaining reasonable accuracy and interpretability. The models used in this study are logistic regression, k-nearest neighbor, decision tree and neural networks and are explained below:

Logistic regression

Logistic regression is used when the results could be treated as a category, and the goal is to predict the probability for the result to fall in that category. Consider a scenario like in this scenario

where the question that arises is “Is this transaction fraudulent or not?” In this scenario we have two categories (fraudulent and not fraudulent) and by using logistic regression, we are able to estimate the probability for that given category.

K-Nearest Neighbors Method & K-Means Method

K-nearest neighbors (KNN) is a machine learning algorithm used for both classification and regression tasks. It operates by utilizing labeled input data to learn a function that can predict the labels or values of new, unlabeled input data. KNN is often referred to as a "lazy" algorithm because it does not perform explicit calculations during the training phase. In KNN, each data point in the training dataset is represented as a vector of features, along with its corresponding label or value. During the prediction phase, when presented with a new, unlabeled data point, the algorithm determines the K nearest neighbors to that point based on a distance metric, typically Euclidean distance. The value or label of the new data point is then determined by a majority vote (for classification) or by averaging (for regression) the labels or values of its K nearest neighbors.

Classification Tree

Classification Tree Analysis (CTA), also known as Decision Tree Analysis, is a machine learning algorithm primarily used for classification tasks and decision-making. It operates by constructing a tree-like model, where the branches represent attributes or features, and the leaves represent decisions or classes. CTA is particularly useful when the task involves classifying data into different categories based on straightforward attributes that can be easily understood, explained, and represented in code. The construction of the classification tree begins with the selection of the most significant attribute that best separates the data based on their class labels. This attribute is placed at the root node of the tree. The subsequent branches of the tree correspond to the possible values of the chosen attribute, leading to different paths or subgroups. This process continues recursively for each subgroup until the tree is built, and the leaves contain the final classification decisions.

Deep Learning (Neural Networks)

Deep learning utilizes neural networks consisting of multiple interconnected layers of nodes, inspired by the human brain's neural structure. Each node, or neuron, processes input data and passes information through weighted connections to subsequent layers. This study will employ a neural network model built using TensorFlow, optimized for identifying complex, nonlinear patterns within transaction data. Neural networks are especially suitable for fraud detection due to their capability to learn intricate patterns from large datasets. By iteratively adjusting weights and biases through backpropagation, the model minimizes prediction errors, enhancing its ability to distinguish between fraudulent and legitimate transactions.

The structure of the paper is as follows. The introduction and background provide an overall application of the machine learning algorithms its use in fraud detection by identifying the transaction patterns. The Machine Learning Phase provides the methods used in building the models and selecting the optimum model. The Application Development Phases describes the integration of machine learning model with database or csv input files as well as application of security best practices in developing an automated system for deploying the best model from the machine learning phase. Finally, the paper concludes with the research results, conclusions, future works and recommendations.

3.0. Machine Learning Phase: Model Development & Evaluation

The goal of this research is to analyze and determine what machine learning model best performs taking into account high accuracy with minimal false negatives and high precision. The method of data sampling and cleaning, exploration data analysis (EDA), model training and comparison, and model selection and hyperparameter tuning.

3.1. Data Description and Model Description

Each variable from the dataset has a significance, for instance, step: 1 step equals 1 hour, type: type of transaction, amount: $ amount of the transaction, nameOrigin: customer starting transaction, oldbalanceOrig: balance before the transaction, newbalanceOrg: balance after the transaction, nameDest: customer receiving the transaction, oldbalanceDestU: initial balance of recipient before the transaction, newbalanceDest: the new balance of recipient after the transaction, isFraud: fraud transaction.

3.2. Exploratory Data Analysis

Initial exploratory analysis revealed that fraudulent activity was highly concentrated in the TRANSFER and CASH_OUT transaction types. These insights guided the development of the preprocessing function, where categorical transaction types were encoded into integer representations. From EDA, it was also noticed that fraudulent transactions tended to involve higher transfer amounts as compared to non-fraudulent ones. In many fraudulent transactions, the oldbalanceDest (original balance of the destination account) was either zero or unchanged, despite large funds being transferred. This suggested that some destination accounts might be dummy or inactive accounts used for fraud, prompting the retention of both oldbalanceDest and newbalanceDest as features. Fraudulent cases accounted for less than 0.2% of all transactions in the original dataset. This class imbalance emphasized the importance of metrics like recall over accuracy. Finally, from EDA, it was noticed that fraudulent and non-fraudulent bank activities could be linearly separable with little outliers. All of these guided the selection of the machine learning algorithms that were used for this study.

3.3. Model Training and Comparison

To determine the best-performing classification model, four machine learning algorithms were evaluated: Logistic Regression, K-Nearest Neighbors (KNN), Decision Tree Classifier, and a Deep Learning model implemented with TensorFlow. Each model was trained and tested on the 80/20 stratified split of the sampled dataset. Evaluation metrics included Recall, Precision, and F1-score, with a primary focus on recall due to the high cost of false negatives in fraud detection contexts.

Table 1: Summary of machine learning models and metrics/scores

Metric	Logistic Regression	KNN	Decision Tree	Deep Learning
Recall	0.44	0.55	**0.71**	0.50
Precision	0.42	0.91	**0.97**	0.91
F1-score	0.43	0.69	**0.82**	0.65

As illustrated in table 1 above, the Decision Tree Classifier outperformed all other models in every key metric, particularly in recall (0.71) and precision (0.97). Its high F1-score of 0.82 indicated a strong balance between sensitivity and specificity. Additionally, its interpretability, quick inference

time, and seamless integration into Scikit-learn's pipeline architecture made it ideal for deployment in the final application phase. The selected Decision Tree model hyperparameters were then further tuned to obtain max_depth parameter = 10, min_samples_split = 2 and min_samples_leaf = 2. These settings offered a trade-off between model complexity and performance, enabling the classifier to generalize well to unseen data without sacrificing fraud detection accuracy.

4.0. Application Development Phase: Pipeline Integration

In the second phase of the project, the selected machine learning model was transformed into a robust, end-to-end application pipeline that can be deployed into a production environment. The main components of the application included data ingestion, security processing, model evaluation and results storage.

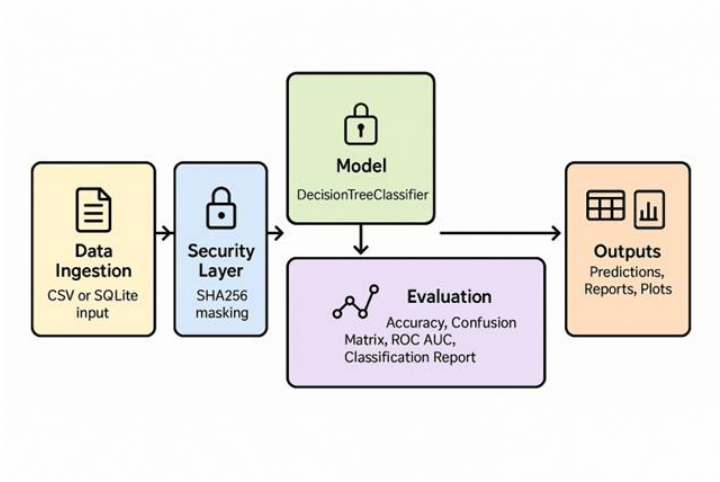


Figure 1: Architecture & Components

The system was designe to handle both CSV file inputs as well as from database (SQLite) tables. This design choice allows the application to seamlessly integrate into different operational environments—whether it's batch processing from raw files or fetching transactions from a relational data store. A security layer is built into the preprocessing stage to ensure sensitive information is protected before any modeling occurs. Specifically, the fields nameOrig and nameDest, which may contain personally identifiable information (PII), are irreversibly hashed using the SHA256 algorithm. After this, the pipeline maps categorical features into numerical features that can be handled by the machine learning algorithm. In the preprocessing step, irrelevant fields are dropped to reduce noise in the learning process. The model component itself uses the DecisionTreeClassifier wrapped inside a Scikit-learn pipeline. This encapsulation simplifies the process of fitting, transforming, and predicting with consistent interfaces across all data formats.

The evaluation module computes essential metrics such as accuracy, recall, F1-score, and the area under the ROC curve. These results are exported as a human-readable text file named model_report.txt. Additionally, a visual representation of the model's performance is created through a ROC curve saved as roc_curve.png. he output phase generates two main result files. For labeled datasets, the application creates a file named fraud_predictions.csv which includes predictions, actual labels, and fraud probability scores. For unlabeled datasets, the output is

stored as fraud_predictions_unlabeled.csv. All output artifacts, including model files, evaluation reports, and plots, are organized within a dedicated Outputs/ subdirectory under the project structure. Lastly, an optional database integration module allows predictions to be written back into a SQLite database. This enhances interoperability with enterprise systems that rely on structured query access for analytics, audits and reporting.

To ensure the robustness, accuracy and security of the application, several unit tests were implemented across multiple components of the system. The application includes a test_pipeline.py module. This is responsible for validating core application logic such as preprocessing transformations, SHA256 masking of sensitive identifiers pipeline construction, and labeled/unlabeled prediction scenarios. It also tests edge cases like missing columns and ensures threshold-based fraud classification behaves as expected. A separate unit test module was developed to test the database connection. test_database.py is dedicated to evaluating the integrity and correctness of database-related scripts. It verifies that the create_db.py and load_csv_to_db.py scripts can correctly generate the required SQLite database schema and populate it with transactional data from CSV sources. These tests ensure that the database interactions conform to expected structure and behavior. End-to-end testing was also performed through command-line execution of the application using various combinations of arguments. These tests validated the program's ability to process data from both file and database sources, generate accurate predictions, and write outputs to the appropriate destinations, including both the file system and the database.

Conclusion And Recommendations

This project presents a robust, end-to-end fraud detection system capable of identifying suspicious banking transactions with high accuracy and interpretability. Beginning with a carefully curated dataset, the machine learning phase involved comprehensive exploratory analysis, feature engineering, model experimentation, and performance benchmarking. Among several evaluated models—including Logistic Regression, K-Nearest Neighbors, and a deep learning neural network—the Decision Tree Classifier emerged as the optimal choice, offering a balance of precision, recall, and interpretability. Its relatively high recall (0.71) and F1-score (0.82) make it particularly effective for minimizing false negatives in a fraud detection context. This was then translated the model into a production-ready application using Python, Scikit-learn, Pandas, and SQLite. The pipeline accepts both CSV and database inputs, applies secure SHA-256 hashing to sensitive fields, preprocesses the data, and outputs predictions, performance reports, and visualization artifacts in a structured Outputs/ directory. Integration with SQLite enables seamless adoption within enterprise data architectures.

There is always room for improvement looking into the future outside of this project. Some suggested future improvements are

- Add web API layer using FastAPI or Flask
- Try out and compare other machine learning models such as XGBoost or LightGBM
- Monitor model drift using MLFlow
- Deploy interactive dashboard with Streamlit
- Automate retraining with scheduling tools like Airflow or Prefect

Appendix - Theoretical Justification and Evaluation

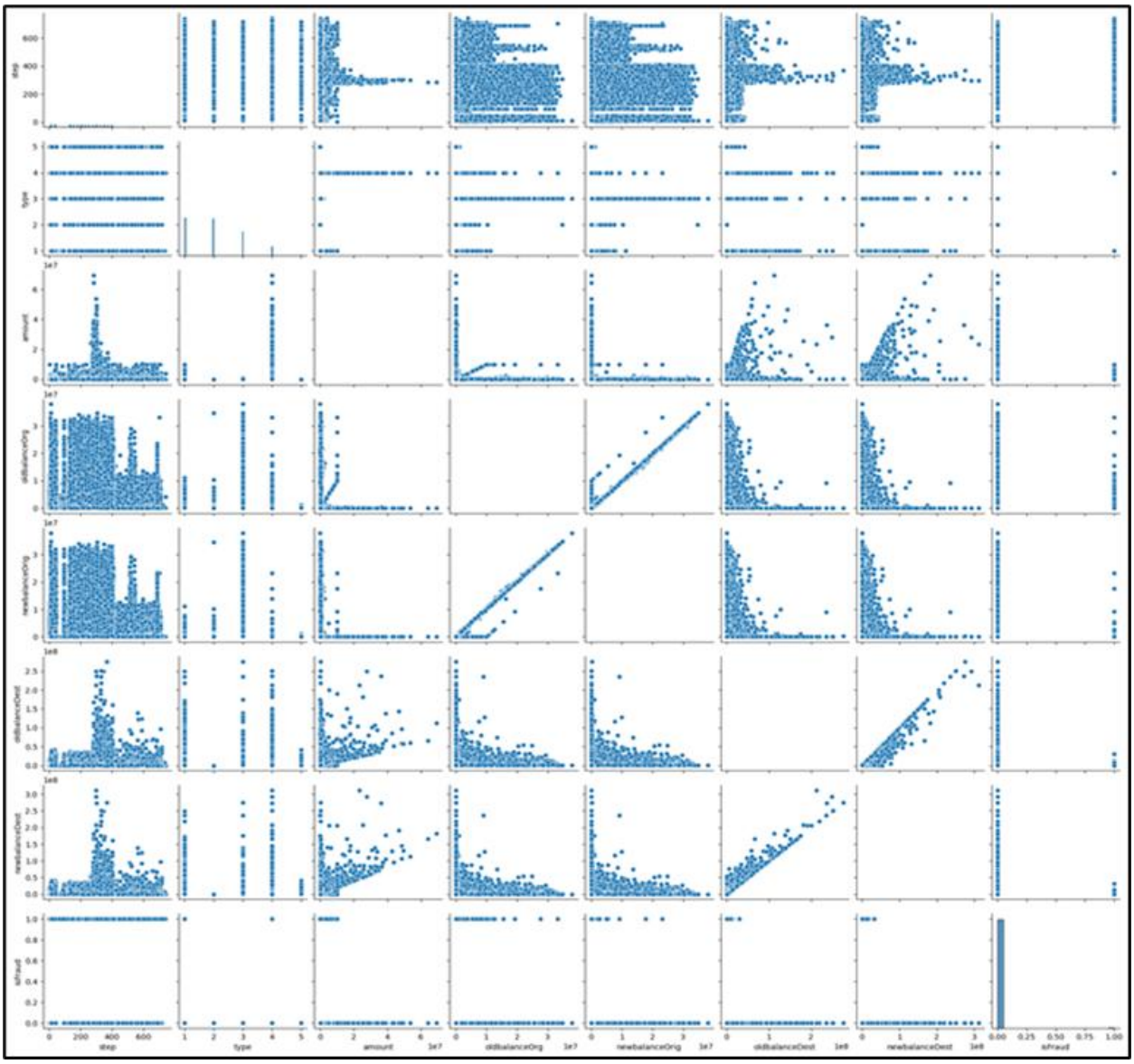
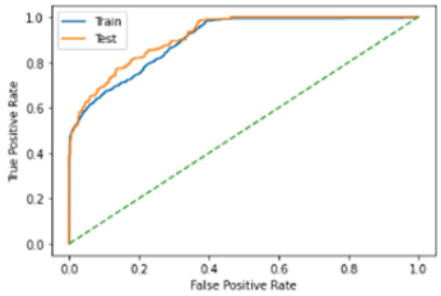


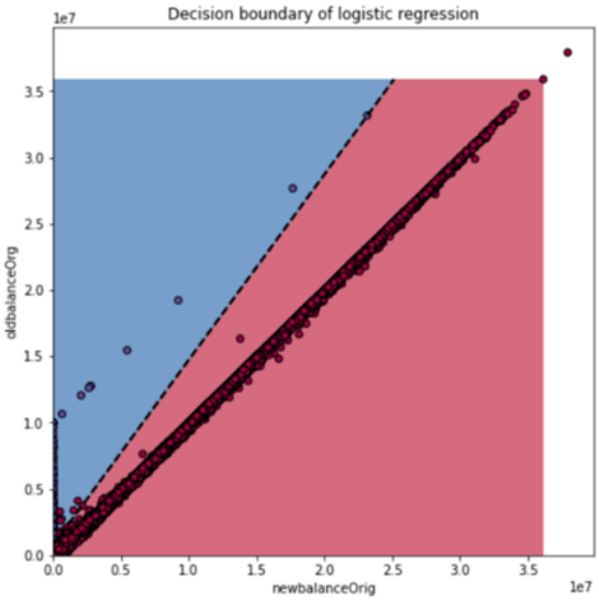
Figure 1: Pearson Correlation

Figure 2: Heat Map

- Training Accuracy : 0.9984458333333334
- Testing Accuracy: 0.9985416666666667

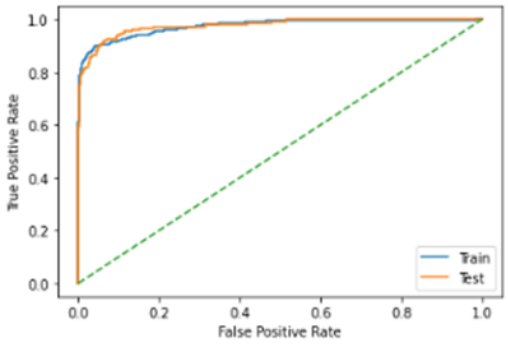
```
Confusion Matrix:
[[119759     91]
 [    84     66]]
------------------------------------------------------------
Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    119850
           1       0.42      0.44      0.43       150

    accuracy                           1.00    120000
   macro avg       0.71      0.72      0.71    120000
weighted avg       1.00      1.00      1.00    120000

Training ROC AUC: 0.899797686882818
Testing ROC AUC: 0.917091753580865
```





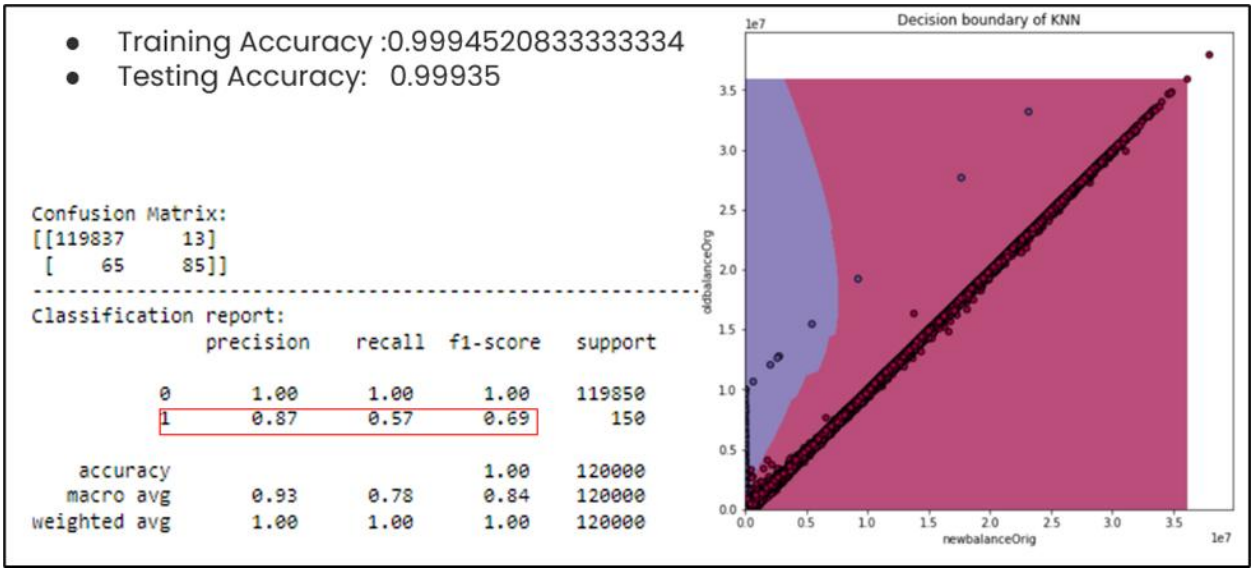
Test Results 1. Logistic Regression results (Default settings)

```
Best hyperparameters: {'solver': 'liblinear', 'penalty': 'l1', 'C': 0.1}
Training accuracy: 99.9310%
Testing accuracy: 99.9275%
Training ROC AUC: 0.9751334499888691
Testing ROC AUC: 0.9772552357113058
```




```
Confusion Matrix:
[[119837     13]
 [    91     59]]
------------------------------------------------------------
Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    119850
           1       0.82      0.39      0.53       150

    accuracy                           1.00    120000
   macro avg       0.91      0.70      0.77    120000
weighted avg       1.00      1.00      1.00    120000
```

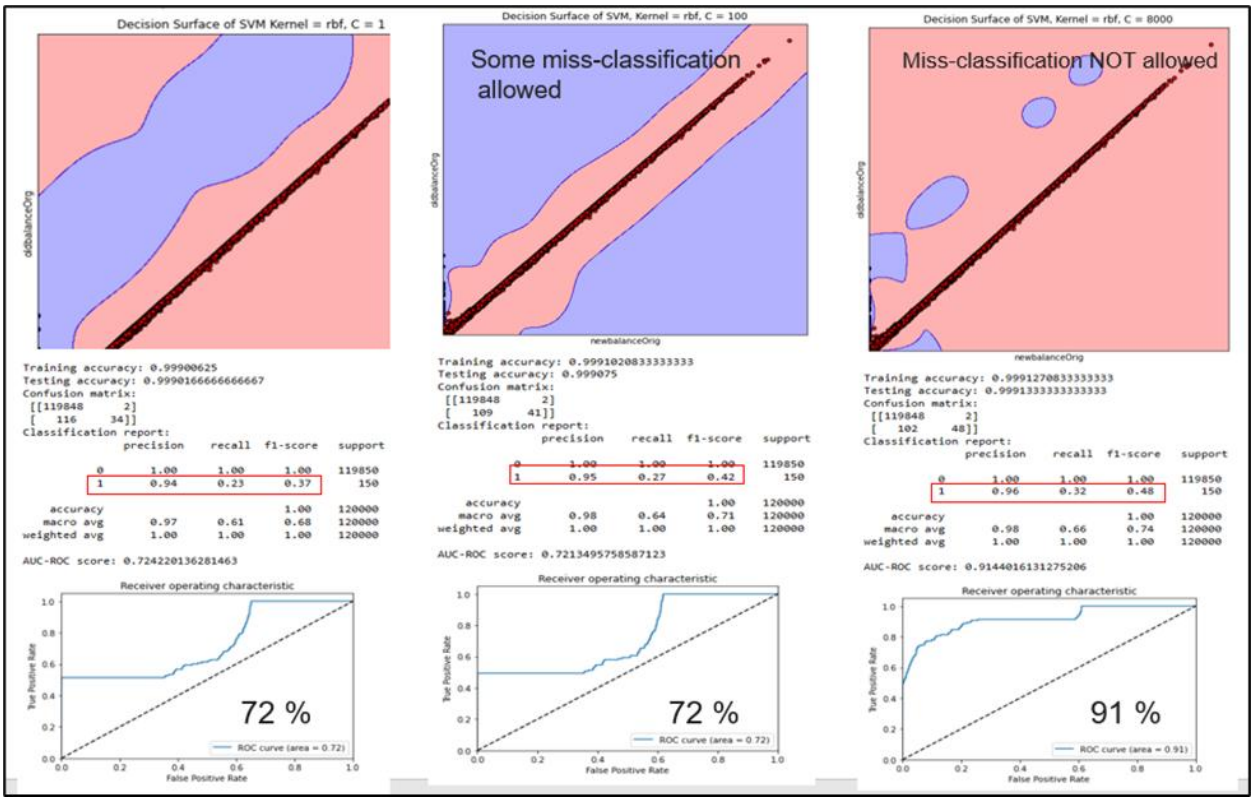
Test Results 2. Logistic Regression results (Tuned settings)



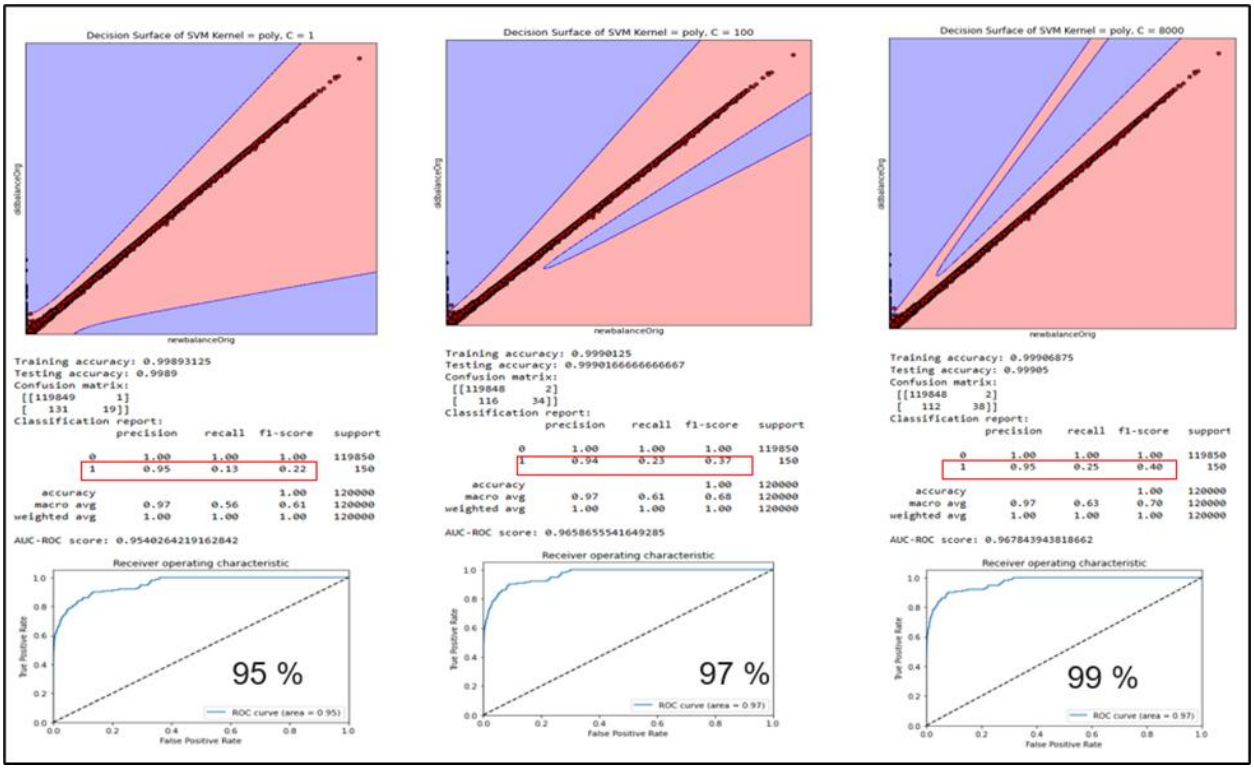
Test Results 3. K-Nearest Neighbor (Default settings)

```
Training accuracy: 1.0
Testing accuracy: 0.999375
Confusion Matrix:
[[119842      8]
 [    67     83]]
--------------------------------------------------------
Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    119850
           1       0.91      0.55      0.69       150

    accuracy                           1.00    120000
   macro avg       0.96      0.78      0.84    120000
weighted avg       1.00      1.00      1.00    120000
```

Test Results 4. K-Nearest Neighbor (Tuned settings)



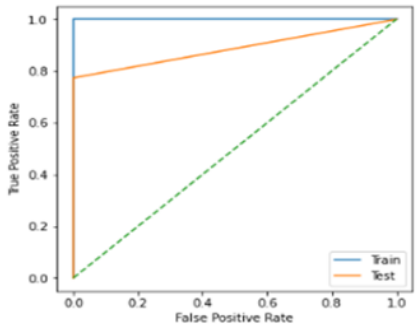
Test Results 5. SVN Kernel-RBF

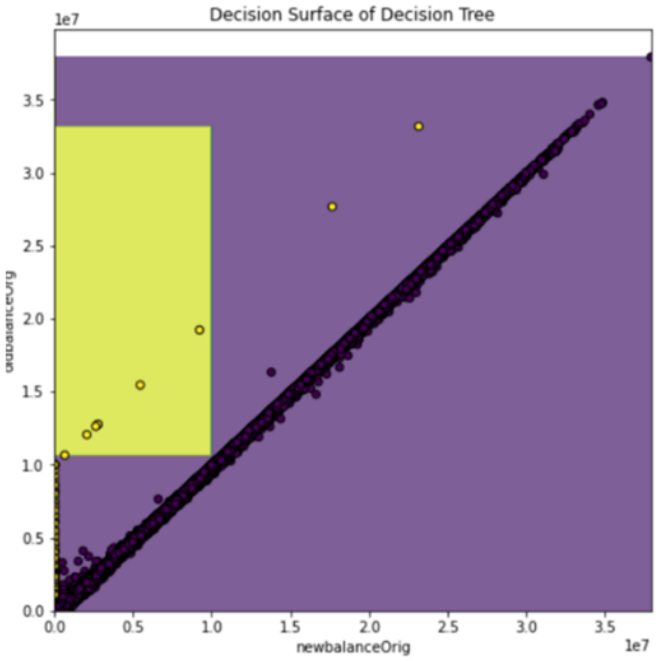


Test Results 6. SVN Kernel - Polynomial

- Training Accuracy : 1.0
- Testing Accuracy: 0.9994916666666667

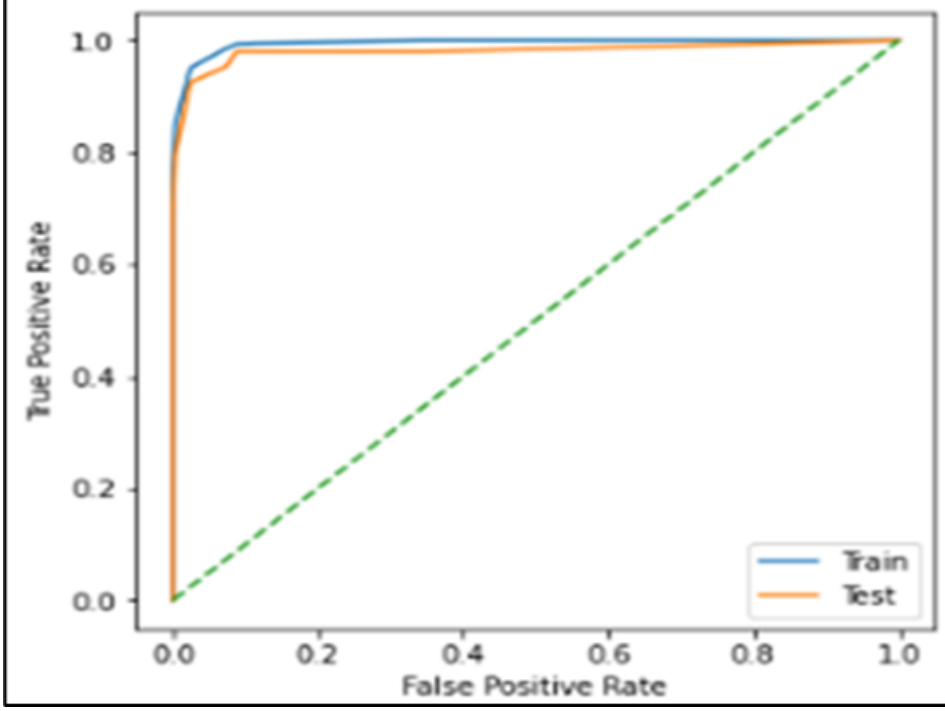
```
Confusion matrix:
 [[119823     27]
 [    34    116]]
Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    119850
           1       0.81      0.77      0.79       150

    accuracy                           1.00    120000
   macro avg       0.91      0.89      0.90    120000
weighted avg       1.00      1.00      1.00    120000

AUC: 0.8865540258656655
```





Test Results 7. Classification tree (Default settings)

```
Training accuracy: 0.999675
Testing accuracy: 0.9996083333333333
AUC: 0.9810015018773468
Confusion matrix:
 [[119847      3]
 [    44    106]]
Classification report:
                precision    recall  f1-score   support

           0       1.00      1.00      1.00    119850
           1       0.97      0.71      0.82       150

    accuracy                           1.00    120000
   macro avg       0.99      0.85      0.91    120000
weighted avg       1.00      1.00      1.00    120000
```



Test Results 8. Classification tree (Tuned settings)