ARTIFICIAL INTELLIGENCE (UCS411)

PROJECT REPORT

BY

Anubhav Gupta (102303789)
Chirag Bansal (102303700)
Shrey Rawat (102303669)
Dushyant Singh (102303807)

GROUP: 2C55

SUBMITTED TO

Dr. Mrinalini Kakroo

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY (A DEEMED TO BE UNIVERSITY)

PATIALA, PUNJAB, INDIA

Second-Year (Jan-May, 2025)

Table of Contents

AI CREDIT CARD FRAUD DETECTION USING ML TECHNIQUES

1. ABSTRACT:

This project explores the application of machine learning techniques for credit card fraud detection using the scikit-learn library. The dataset, obtained from Kaggle, contains transaction records labeled as either fraudulent or legitimate. The primary goal of this study is to develop and compare multiple classification models to effectively identify fraudulent activities in financial transactions. Four distinct algorithms—Logistic Regression, Decision Tree, Random Forest, and k-Nearest Neighbors (k-NN)—were implemented and evaluated. The project focuses on the crucial aspects of data preprocessing, including handling class imbalance through techniques such as oversampling and feature scaling, as well as the importance of selecting appropriate evaluation metrics, such as precision, recall, and F1 score, for imbalanced datasets. Through these techniques, the project aims to provide a comprehensive approach to fraud detection in the context of credit card transactions, emphasizing the critical role of model selection, tuning, and feature engineering in achieving reliable performance.

2. INTRODUCTION:

Credit card fraud detection is a critical application of machine learning in the financial industry, where the goal is to identify fraudulent transactions and prevent significant financial losses. With the increasing volume of online transactions and the sophistication of fraud techniques, automated detection systems are essential for real-time fraud prevention. This project focuses on binary classification for credit card fraud detection using four widely-used machine learning

algorithms—Logistic Regression, Decision Tree, Random Forest, and k-Nearest Neighbors (k-NN)—implemented using the scikit-learn library. The dataset, sourced from Kaggle, contains transaction records that are labeled as either legitimate or fraudulent, providing a challenging task due to the class imbalance inherent in real-world transaction data. The project covers the complete machine learning pipeline, including data preprocessing, feature scaling, model training, evaluation, and result visualization. By comparing different models, the goal is to demonstrate the strengths and weaknesses of each algorithm in detecting fraudulent transactions, with a focus on ensuring robust performance in the face of imbalanced data.

3. PROBLEM STATEMENT:

The primary challenge addressed in this project is building an effective credit card fraud detection system using machine learning algorithms. The goal is to develop models that can accurately distinguish between legitimate and fraudulent transactions in a real-world, imbalanced dataset. Specifically, we aim to:

- Detect fraudulent credit card transactions from a large dataset of transaction records.
- Address the challenges posed by class imbalance using techniques like Synthetic Minority Over-Sampling Technique (SMOTE).
- Analyse and compare the performance of Logistic Regression, Decision Tree, Random Forest, and k-Nearest Neighbors (k-NN).
- Visualize results using performance metrics such as precision, recall, F1 score, and ROC-AUC to evaluate model effectiveness.

4. OBJECTIVES:

- To preprocess and scale transaction data for use in machine learning models, ensuring that features are optimized for model training.

- To handle class imbalance effectively using techniques such as oversampling and synthetic data generation to improve model performance.

- To implement and evaluate four machine learning classifiers—Logistic Regression, Decision Tree, Random Forest, and k-Nearest Neighbors (k-NN)—for credit card fraud detection.

- To assess model performance using key metrics like precision, recall, F1 score, and ROC curve, ensuring that models are optimized for detecting fraudulent transactions.

- To visualize model results with confusion matrices, precision-recall curves, and feature importance charts to provide deeper insights into model behavior.

5. METHODOLOGY:

- **DATASET:**

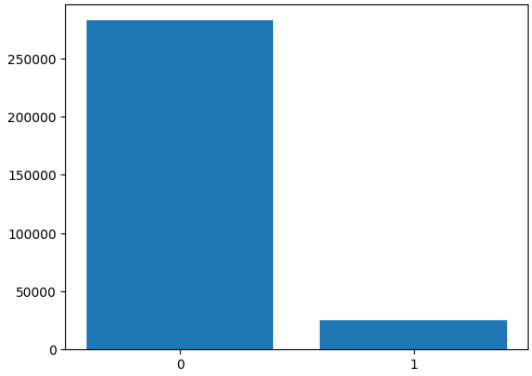
  We used the Credit Card Fraud Detection dataset from Kaggle, which contains transaction records labelled as either fraudulent or legitimate. The dataset consists of anonymized transaction features, including details such as the transaction amount, total income, and various other features required for classification.
  The dataset is highly imbalanced, with a much smaller proportion of fraudulent transactions compared to legitimate ones. It is split into training and test sets, allowing for model development and evaluation. Each record includes features that are crucial for detecting fraudulent activity, making it suitable for building a binary classification model.

- **PREPROCESSING:**

Data preprocessing is a crucial step in any machine learning pipeline, particularly for real-world datasets like credit card fraud detection, where data quality and structure can significantly impact model performance. In this project, we followed a series of steps to preprocess the raw transaction data, ensuring that it was ready for use in machine learning models. The main preprocessing steps included:

1. Checking Data Imbalance

One of the primary challenges in fraud detection is the class imbalance in the dataset, where fraudulent transactions are far less frequent than legitimate ones. To assess the extent of the imbalance, we visualized the distribution of the target variable (fraud vs. legitimate) using bar plots. This imbalance can negatively affect model performance, particularly for algorithms that are sensitive to the majority class. In this project, we took steps to address this issue by applying techniques like oversampling (using methods like SMOTE) and adjusting class weights during model training to mitigate its impact.



2. Manually Selecting Features

Not all features in the dataset contribute equally to the prediction of fraudulent transactions. To improve model performance and reduce overfitting, we manually selected a subset of relevant features. This process involved examining the domain knowledge and data understanding to identify which features were most likely to help in distinguishing fraudulent from legitimate transactions. We avoided using irrelevant or redundant features, aiming to keep

the model as simple and interpretable as possible while retaining the most important information for classification.

```
['TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'AMT_INCOME_TOTAL',
 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE','DAYS_BIRTH','DAYS_EMPLOYED', 'NAME_EDUCATION_TYPE',
 'OCCUPATION_TYPE','OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE','AMT_REQ_CREDIT_BUREAU_DAY'
 ,'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CREDIT_BUREAU_MON','CNT_FAM_MEMBERS',
 'REGION_RATING_CLIENT', 'REG_REGION_NOT_WORK_REGION', 'DAYS_LAST_PHONE_CHANGE']
```

3. Removing Null Values

Handling missing data is an important part of the preprocessing pipeline. In our dataset, some records contained missing or null values for certain features. We removed rows with null values using the df.dropna() method to ensure that the data used for training and testing did not include incomplete records. This step ensured that our models were trained on a clean dataset, which is critical for achieving reliable predictions.

4. Encoding String Categorical Features

Some of the features in the dataset were categorical, stored as string values (e.g., transaction types or country of origin). To make these features suitable for machine learning algorithms, we encoded them into numerical representations. For this, we applied label encoding or one-hot encoding depending on the number of categories in each feature. Label encoding was used for ordinal features, where there is a natural ordering of values, while one-hot encoding was applied to nominal features, where no such ordering exists. This transformation enabled us to use categorical data effectively in the machine learning models.

5. Applying SMOTE

To address the class imbalance further, we applied SMOTE (Synthetic Minority Over-sampling Technique) to generate synthetic samples for the minority class (fraudulent transactions). SMOTE works by selecting a minority class sample and creating synthetic samples along the line segments connecting the sample to its nearest neighbors. This increased the number of fraudulent transaction samples and helped mitigate the risk of the model being biased towards predicting legitimate transactions. The application of SMOTE was done after feature selection and before splitting the data into training and testing sets.

6. Splitting the Data into Training and Testing

Finally, the dataset was split into training and testing sets to evaluate model performance. We used an 80-20 split, where 80% of the data was allocated to training and 20% to testing. This division ensured that the model could be trained on a substantial amount of data while leaving enough unseen data for testing and performance evaluation. The splitting was done after applying SMOTE to ensure that both the training and testing sets reflected the same class distribution.

- **MODEL IMPLEMENTATION:**

  The core of this project involves applying and evaluating several machine learning algorithms for credit card fraud detection. The models used in this project include Logistic Regression, Decision Tree, Random Forest, and k-Nearest Neighbors (k-NN), all implemented using the scikit-learn library. Using scikit-learn allows for leveraging highly optimized implementations of these algorithms while maintaining the flexibility to experiment with different model configurations and hyperparameters.

1. Logistic Regression

Logistic Regression is a binary classification algorithm that predicts the probability of a class membership using the sigmoid function. A One-vs-Rest (OvR) approach is employed to extend Logistic Regression for multi-class classification in this project.

Training Phase:

- The class_weight='balanced' parameter is set to automatically adjust weights for imbalanced classes, making the model more sensitive to the minority class (fraudulent transactions).
- The max_iter=1000 parameter is set to ensure that the model converges to a solution, allowing up to 1000

iterations during training to refine the coefficients.

Prediction Phase:

- After training, the model uses the sigmoid function to predict the class probabilities, and the class with the highest probability is selected as the predicted label.

2. Decision Tree Classifier

The Decision Tree algorithm builds a tree structure by recursively splitting the data based on feature values, creating nodes that lead to a decision (leaf). It is widely used for classification due to its interpretability and ability to handle both categorical and continuous data.

Training Phase:

- max_depth=15 is set to limit the depth of the tree and prevent overfitting. This controls how complex the tree can become, balancing bias and variance.
- class_weight='balanced' ensures that the tree handles imbalanced classes effectively by giving more weight to the minority class (fraudulent transactions).
- random_state=45 is set to ensure reproducibility of the results.

Prediction Phase:

- The tree classifies a test instance by traversing from the root to a leaf based on the feature values of the instance.

3. Random Forest Classifier

Random Forest is an ensemble method that combines multiple decision trees trained on random subsets of the data and features. It helps reduce overfitting and improves model robustness.

Training Phase:

- n_estimators=100 sets the number of decision trees in the forest. With 100 trees, the model averages the predictions of all trees to produce a final result.
- max_depth=15 limits the depth of each individual tree in the forest, helping prevent overfitting while maintaining the model's ability to capture complex patterns.
- class_weight='balanced' ensures the model handles the imbalanced dataset by giving higher weight to the minority class (fraudulent transactions).
- random_state=45 ensures that the training process is reproducible.

Prediction Phase:

- The model uses majority voting to aggregate the predictions of all trees, assigning the most frequent class as the predicted label.

4. k-Nearest Neighbors (k-NN)

The k-NN algorithm makes predictions based on the closest neighbors to a test instance in the feature space. It is a simple, instance-based learning method where the model memorizes the training data and performs prediction based on similarity.

Training Phase:

- n_neighbors=5 sets the number of neighbors to consider when making predictions. This parameter helps determine how many nearby instances influence the predicted class for a test instance.
- weights='uniform' means that each of the k neighbors contributes equally to the prediction. If "distance" is used, closer neighbors would have a larger influence, but here the uniform weighting ensures that each neighbor contributes equally.
- n_jobs=-1 ensures that the model utilizes all available cores on the machine for parallel computation, speeding up the process, particularly when the dataset is large.

Prediction Phase:

- The model computes the distances to all training instances and selects the k nearest neighbors. The predicted class is determined by majority voting among the k neighbors.

Each model was trained using the specified hyperparameters to ensure optimal performance in predicting fraudulent transactions. The models were evaluated using various metrics, such as accuracy, precision, recall, F1 score, and ROC-AUC, with special attention given to the class imbalance inherent in fraud detection problems. Additionally, the results were visualized using confusion matrices and precision-recall curves to gain deeper insights into model performance.

- **EVALUATION:**

Evaluating machine learning models is a critical part of understanding their performance and assessing how well they generalize to unseen data. In this project, the models (Logistic Regression, Decision Tree, Random Forest, and k-Nearest Neighbors) were evaluated using multiple performance metrics and visual tools to compare their effectiveness, especially in the context of class imbalance in fraud detection. The evaluation was performed on the test split of the dataset, which contains instances that were not seen during training.

1. Classification Report

To gain an overview of the performance of each classifier, we used the classification report from scikit-learn. The classification report provides a detailed summary of the model's precision, recall, F1-score, and support (the number of occurrences of each class). These metrics give a clearer understanding of how the model performs for each class, which

is crucial for imbalanced datasets.

For example, for **Random Forest**:

```
y_pred_rf = r_f.predict(x_test)
print(classification_report(y_test, y_pred_rf))
```

This report helps to evaluate:

- **Precision**: The proportion of positive predictions that are actually correct.
- **Recall**: The proportion of actual positives that are correctly identified.
- **F1-Score**: The harmonic mean of precision and recall, providing a single metric that balances both.
- **Support**: The number of occurrences of each class in the test data.

2. Confusion Matrix

A confusion matrix provides a comprehensive view of how well the model is performing across all classes. It shows the number of correct and incorrect predictions for each class, allowing us to analyze the performance in greater detail. The confusion matrix helps to:

- Identify which classes are most often confused by the model.
- Highlight if the model has a bias toward certain classes.
- Show where the model performs well and where it struggles.

We visualized the confusion matrix using a seaborn heatmap, which clearly displayed where the model made errors and where it performed well.

3. ROC Curve

The Receiver Operating Characteristic (ROC) curve and Area Under the Curve (AUC) are key metrics for evaluating classifier

performance, especially for binary classification tasks such as fraud detection. The ROC curve plots the true positive rate (sensitivity) against the false positive rate (1 - specificity), and the AUC score provides a summary measure of the classifier's ability to distinguish between the classes.

The **AUC** value helps to determine the overall effectiveness of the model:

- A model with an AUC close to 1 indicates that it performs well.
- An AUC around 0.5 suggests that the model's performance is similar to random guessing.

4. Precision-Recall Curve

Given the class imbalance present in fraud detection, the Precision-Recall Curve is often a more informative evaluation metric than the ROC curve. It helps us understand the trade-off between precision and recall at different thresholds.

- **Precision** measures how many of the predicted positive instances are actually positive.
- **Recall** measures how many of the actual positive instances were correctly identified.
- The curve shows the trade-off between these two metrics, helping to choose the optimal threshold for classification.

5. Hyperparameter Tuning

To enhance the performance of each model, hyperparameter tuning was performed to identify the optimal set of parameters. This process involved testing various combinations of hyperparameters and selecting the ones that yielded the best performance on the validation set. For each model, parameters like the number of estimators, maximum depth, and regularization strength were fine-tuned using grid search or randomized search techniques. This tuning was crucial in improving the models' ability to classify fraudulent transactions more accurately, as it helped to balance bias and variance for each

algorithm.

6. RESULTS:

The performance of the models was evaluated using several metrics, including accuracy, precision, recall, F1-score, and AUC.

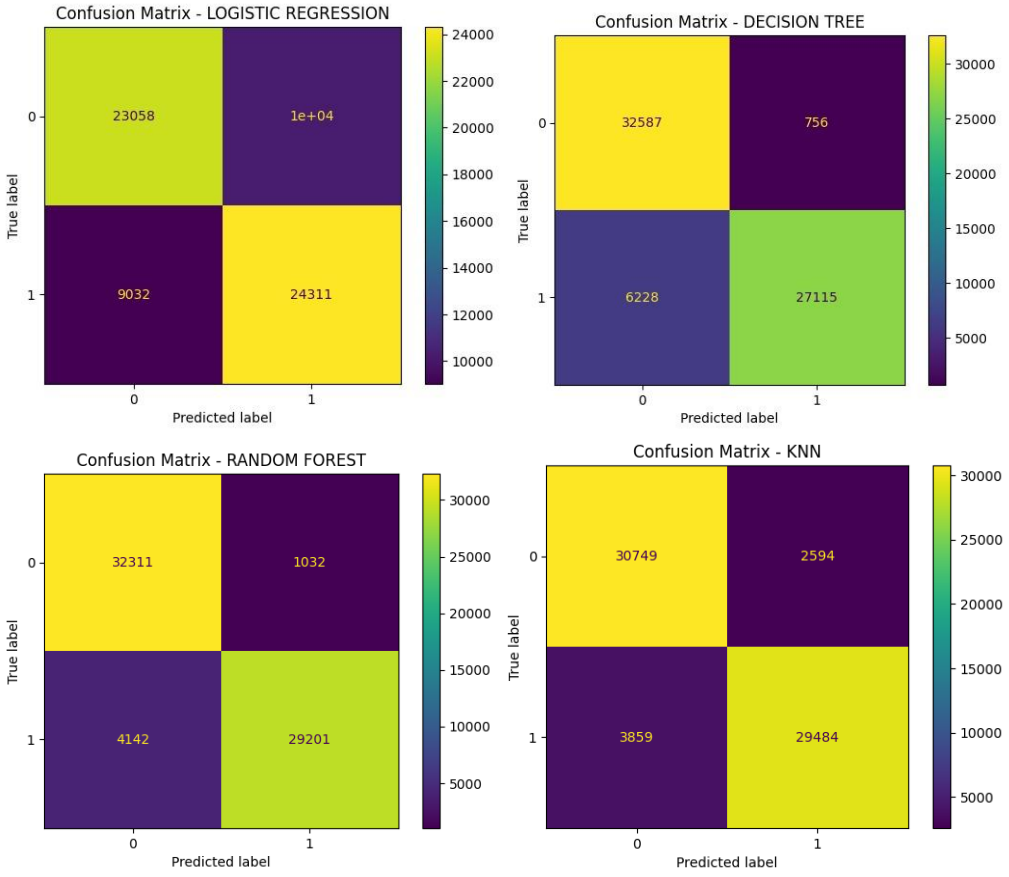
1. Model Performance

- **Decision Tree:**
  - Accuracy: 90%
  - Precision (fraudulent): 0.97, Recall (fraudulent): 0.81
  - The Decision Tree model performed well in terms of recall for non-fraudulent transactions (0.98), but its recall for fraud detection was lower (0.81). It showed good precision for fraudulent cases but had some trade-off with recall.
- **Logistic Regression:**
  - Accuracy: 71%
  - Precision (fraudulent): 0.70, Recall (fraudulent): 0.73
  - Logistic Regression performed decently but struggled to balance precision and recall for fraud detection. It had the lowest accuracy and was less effective at identifying fraudulent transactions compared to the other models.
- **k-NN:**
  - Accuracy: 90%
  - Precision (fraudulent): 0.92, Recall (fraudulent): 0.88
  - The k-NN model performed similarly to the Decision Tree, achieving high precision and recall for both fraudulent and non-fraudulent transactions. It showed a strong ability to detect fraudulent transactions with high accuracy (90%).
- **Random Forest:**
  - Accuracy: 92%
  - Precision (fraudulent): 0.97, Recall (fraudulent): 0.88
  - The Random Forest model demonstrated the highest accuracy (92%) among all models, with strong precision (0.97) and good recall (0.88) for fraudulent transactions.

This indicates its excellent performance in classifying both fraud and non-fraud cases.

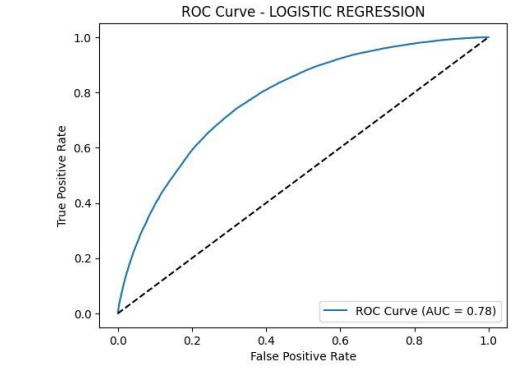
2. CONFUSION MATRIX:

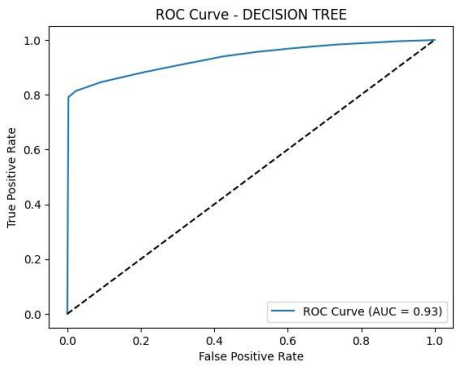
Below are the confusion matrices for each of the models used in this project, which visually represent the classification results.

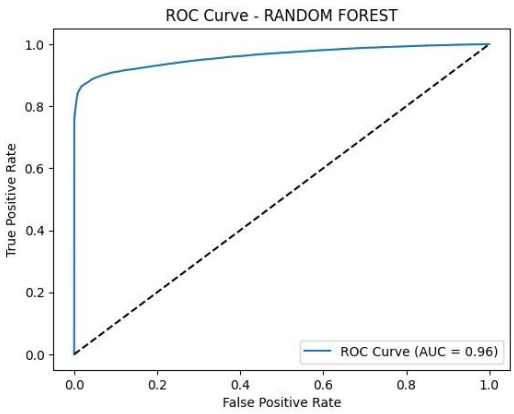


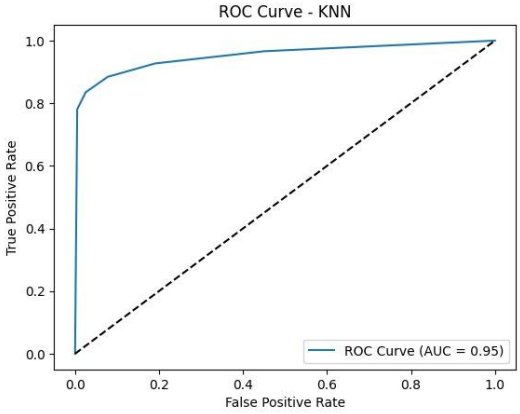
3. ROC CURVE:

The AUC (Area Under the Curve) is a crucial metric, as a higher AUC value indicates a better-performing model. Below are the ROC curves for each of the models, showcasing how effectively they separate fraudulent transactions from non-fraudulent ones.



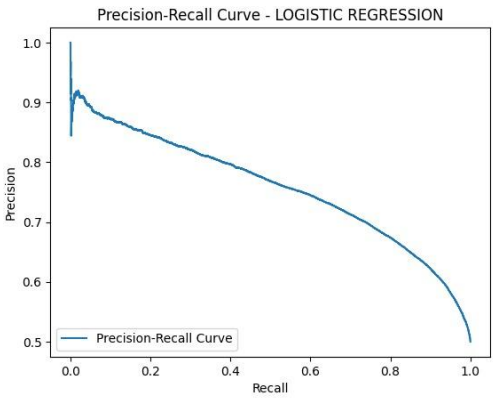


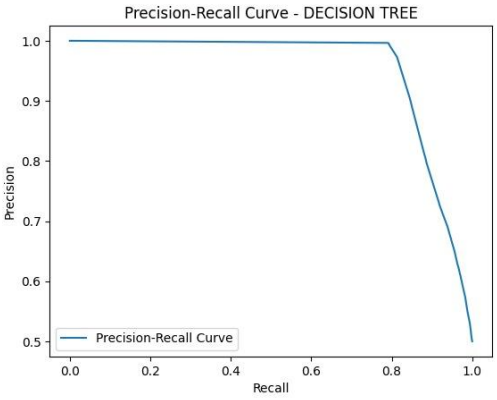


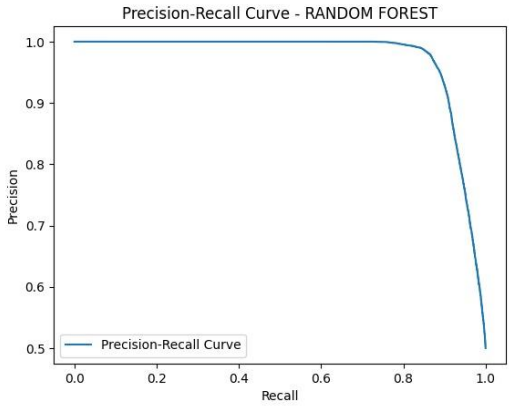


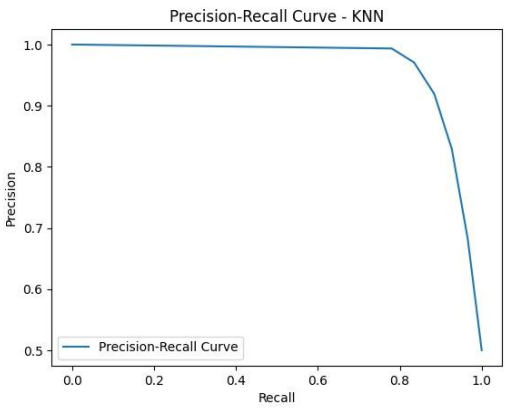
4. PRECISION-RECALL CURVE:

This curve plots Precision (the proportion of true positives among all predicted positives) against Recall (the proportion of true positives among all actual positives). A higher curve indicates better performance, particularly for the minority class. Below are the Precision-Recall curves for each of the models, highlighting their ability to correctly identify fraudulent transactions while minimizing false positives.









5. HYPERPARAMETER TUNING:

By tuning parameters such as the number of estimators, depth of trees, regularization strength, and others, the models were able to achieve better performance in classifying fraudulent transactions. Below are the results of the hyperparameter tuning, which include the best parameters and the corresponding performance scores for each model.

	model	best_score	best_params
0	decision_tree	0.915027	{'max_depth': 25}
1	random_forest	0.936740	{'max_depth': 20, 'n_estimators': 100}
2	logistic_regression	0.582882	{'max_iter': 1000, 'solver': 'liblinear'}
3	knn	0.922030	{'knn__n_neighbors': 9, 'knn__p': 1, 'knn__wei...

7. CONCLUSION:

This project evaluated four machine learning algorithms—Logistic Regression, Decision Tree, Random Forest, and k-Nearest Neighbors (k-NN)—for credit card fraud detection. After preprocessing the data to address class imbalance and scaling features, each model was trained and evaluated using various performance metrics.

The Random Forest model achieved the highest accuracy (92%) and strong precision and recall for detecting fraud, making it the best performer. k-NN and Decision Tree also showed good results with 90% accuracy, though the Decision Tree was better at detecting non-fraudulent transactions. Logistic Regression had the lowest accuracy (71%) and struggled with balancing precision and recall for fraud detection.

Overall, the evaluation metrics and visualization tools like confusion matrices and precision-recall curves highlighted the strengths and weaknesses of each model. The results demonstrate the effectiveness of machine learning in fraud detection, with Random Forest standing out as the most reliable model. Future work could explore ensemble methods, cost-sensitive learning, or deep learning techniques for further improvement.

8. REFERENCES:

1. Credit Card dataset
https://www.kaggle.com/datasets/mishra5001/credit-card/data

2. For Logistic Regression
https://www.geeksforgeeks.org/implementation-of-logistic-regression-from- scratch-using-python/

3. Decision Tree
https://www.geeksforgeeks.org/decision-tree/

4. Random Forest
https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/

5. KNN (k-Nearest Neighbors)
https://www.geeksforgeeks.org/k-nearest-neighbours/

6. Hyperparameter Tuning
https://www.geeksforgeeks.org/hyperparameter-tuning/

7. Confusion Matrix and ROC curve
https://statisticallyrelevant.com/confusion-matrix-and-roc-curves/

8. Precision Recall Curve
https://www.datacamp.com/tutorial/precision-recall-curve-tutorial