Fraud Detection in Credit Card Transactions using Machine Learning

By: Aarohi Garg
Submitted To: Prof Thanh Phuong Nguyen
Date: 17 Jan 2024

Abstract

This project tackles credit card fraud detection using machine learning and deep learning, focusing on the challenge of class imbalance in transaction data. It starts with an Exploratory Data Analysis (EDA) and feature engineering, including scaling and categorical encoding. To address class imbalance, the study explores oversampling and under sampling techniques. It then evaluates three machine learning and three deep learning models based on precision, recall, and ROC curve metrics. This comparative analysis provides insights into effective methods for fraud detection.

Dataset

The dataset used for this project is: Credit Card Transactions Fraud Detection Dataset (link)

This is a simulated credit card transaction dataset containing legitimate and fraud transactions from the duration 1st Jan 2019 - 31st Dec 2020. It covers credit cards of 1000 customers doing transactions with a pool of 800 merchants.

This was generated using Sparkov Data Generation | Github tool created by Brandon Harris. This simulation was run for the duration - 1 Jan 2019 to 31 Dec 2020. The files were combined and converted into a standard format.

This simulated dataset is based on another Credit Card Fraud Detection Dataset (link).

The original dataset contains transactions made by credit cards in September 2013 by European cardholders. It presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

The training set for Credit Card Transactions contains the following fields:

- index - Unique Identifier for each row
- trans_date_trans_time - Transaction DateTime
- cc_num - Credit Card Number of Customer
- merchant - Merchant Name
- category - Category of Merchant
- amt - Amount of Transaction
- first - First Name of Credit Card Holder
- last - Last Name of Credit Card Holder
- gender - Gender of Credit Card Holder
- street - Street Address of Credit Card Holder
- city - City of Credit Card Holder
- state - State of Credit Card Holder
- zip - Zip of Credit Card Holder
- lat - Latitude Location of Credit Card Holder
- long - Longitude Location of Credit Card Holder
- city_pop - Credit Card Holder's City Population
- job - Job of Credit Card Holder
- dob - Date of Birth of Credit Card Holder
- trans_num - Transaction Number
- unix_time - UNIX Time of transaction
- merch_lat - Latitude Location of Merchant
- merch_long - Longitude Location of Merchant

- is_fraud - Fraud Flag <--- Target Class

Exploratory Data Analysis (EDA)

Observation 1

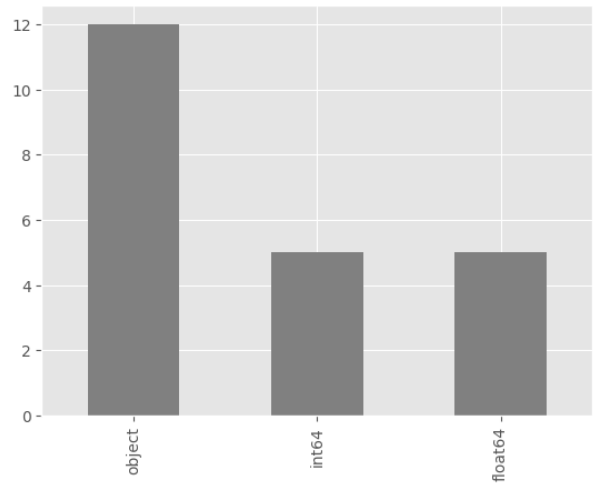


Figure 1 - Datatype of Fields

From Figure 1, we can see that there are a lot of categorical features which will require categorical encoding.

Observation 2

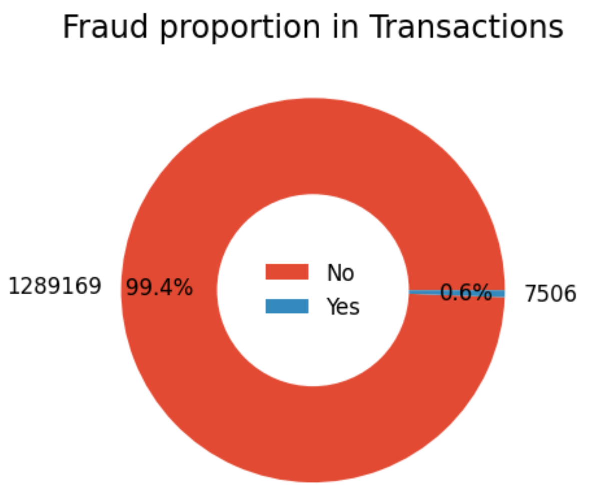


Figure 2 - Fraud proportion in Transactions

From Figure 2, it can be clearly seen that the dataset is highly imbalanced due to which Oversampling/Undersampling techniques have to implemented while model training.

Observation 3

In Figure 3, we can see that most of the fraudulent transactions occur between the amount of 0 to 100.

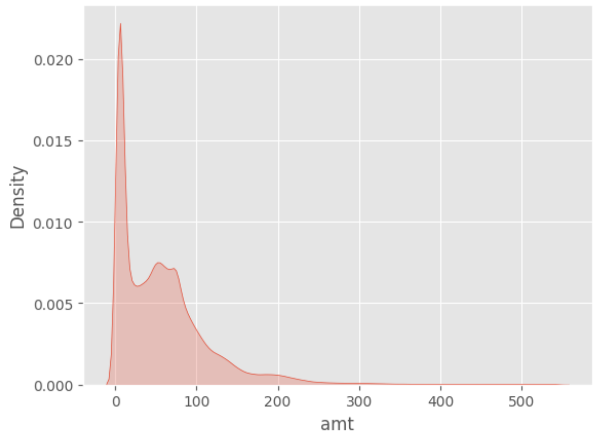


Figure 3 - Distribution of Amount Feature

Determining Categories prone to Frauds

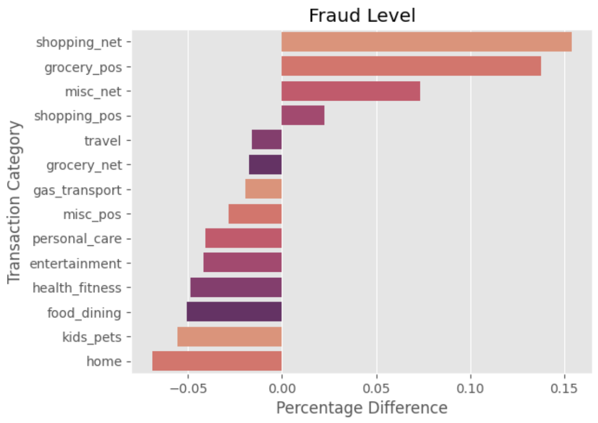


Figure 4 - Fraud Level of different categories

The categories with percentage difference greater than 0 and more prone to frauds, namely: *shopping_net, grocery_pos, misc_net, shopping_pos*.

Impact of Age on Fraudulent Transactions

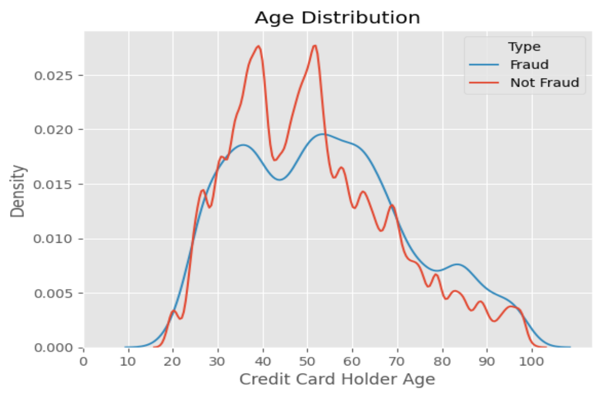


Figure 5 - Age Distribution

It can be seen that in *non-fraudulent transactions*, there is a peak around **35 - 40** years, and another one around **50** years. On the other hand, In *fraudulent* operations, we observe a smoother distribution, with peaks around **35** years and in the range of **50 - 55** years.

Impact of Time on Fraudulent Transactions

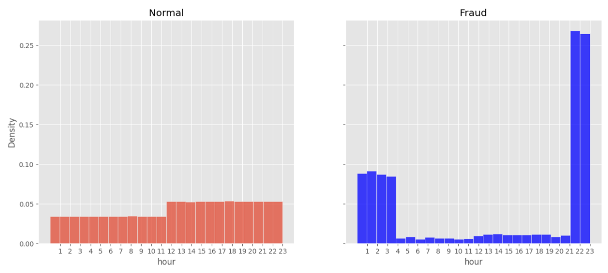


Figure 6 - Time Distribution

There is a clear pattern when it comes to hour in the day. Fraudulent payments happens more frequently around midnight than in normal transactions.

Impact of Month on Fraudulent Transactions

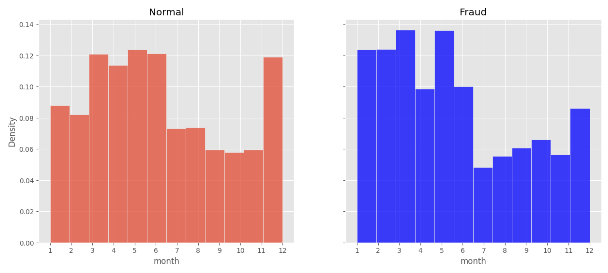


Figure 7 - Month Distribution

There is no pattern in fraud transactions with respect to the month.

Feature Engineering

Scaling

Scaling is a pre-processing technique used to standardize the range of independent variables or features of data. It's essential because different features often have different scales, which can create biases in the training of machine learning models.

From Figure 3, we can see that "amt" category has very little spacing between small numbers and large spacing between high numbers. To tackle this, we use **Logarithmic scaling (log1p)**.

The importance of log1p is that the function $\log(1 + x)$ is well-conditioned near zero, and often turns up in numerical algorithms or algebraic rearrangements of other functions.

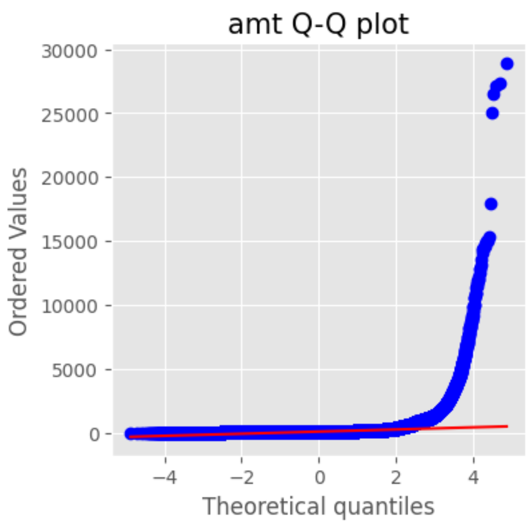


Figure 8 - Normality of 'amt' before scaling

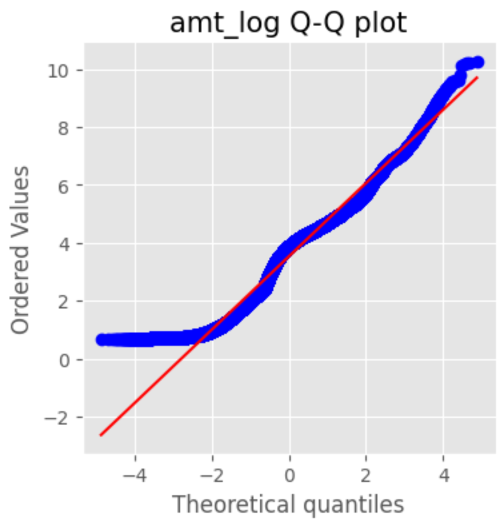


Figure 9 - Normality of 'amt' after scaling

After performing logrithmic scaling, the skewness of the feature improved a lot.

Skewness is a statistical term and it is a way to estimate or measure the shape of a distribution used to estimate the asymmetrical behavioural. A skewness value greater than zero means that there is more weight in the right tail of the distribution.

Categorical Encoding

Categorical encoding is a process in machine learning where categorical data, which are often text labels, are converted into numerical form. This transformation is essential because most machine learning algorithms can only interpret numerical inputs.

For this purpose we have used **Weight of Evidence Encoder (WOE)**. It quantifies the strength of the relationship between a categorical independent variable (predictor) and a binary target variable (response) by calculating the *logarithm of the odds ratio*. It measures how well the category predicts the positive (1) or negative (0) class of the target variable.

- If WOE>0, it indicates that the category is associated with a higher likelihood of the positive event (good outcome).
- If WOE<0, it indicates that the category is associated with a higher likelihood of the negative event (bad outcome).
- If WOE=0, it suggests that the category has no discriminatory power between the positive and negative events.

Oversampling Techniques

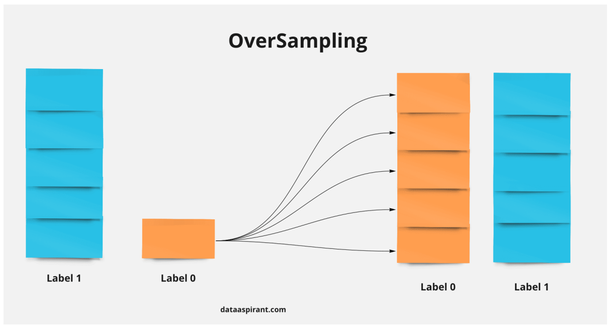


Figure 10 – Oversampling

Oversampling in machine learning is a technique used to address class imbalance in datasets, where one class significantly outnumbers another. It involves artificially augmenting the minority class by replicating its samples or generating new synthetic samples. This is crucial because class imbalance can lead to biased models that perform well on the majority class but poorly on the minority class, which is often of greater interest (e.g., in fraud detection). Oversampling aims to balance the class distribution, ensuring that the model learns equally from both classes, thereby improving its performance and generalization ability on underrepresented classes.

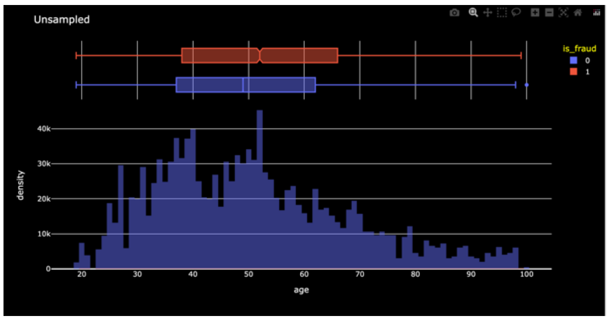


Figure 11 - Unsampled Dataset

From Figure 11, we can see that without sampling, the dataset hardly contains any fraudulent transactions compared to the non-fraud ones.

SMOTE

SMOTE (Synthetic Minority Over-sampling Technique) is a method used to address class imbalance in machine learning datasets. It generates synthetic samples for the minority class by interpolating between existing minority instances. SMOTE selects a minority class instance and finds its nearest neighbours, then creates new instances that combine features of the chosen instance with its neighbours, leading to a more balanced dataset.

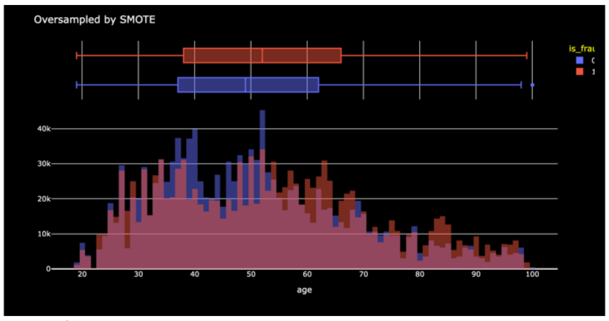


Figure 12 - Oversampled by SMOTE

RandomOverSampler

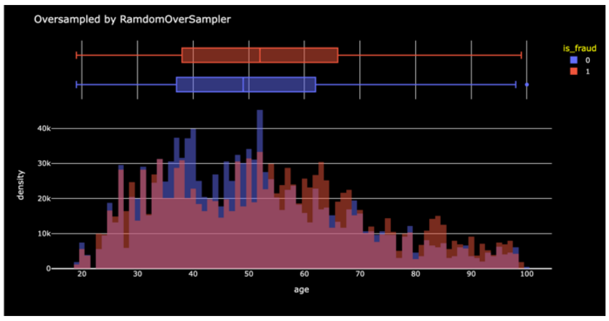


Figure 13 - Oversampled by RandomOverSampler

Random oversampling involves randomly duplicating examples from the minority class and adding them to the training dataset.

ADASYN

ADASYN (Adaptive Synthetic Sampling) is an oversampling technique used in machine learning to handle class imbalance. It generates synthetic samples for the minority class, focusing more on generating samples near the borderline where misclassification is more likely. This approach improves learning about the decision boundary, enhancing classifier performance.

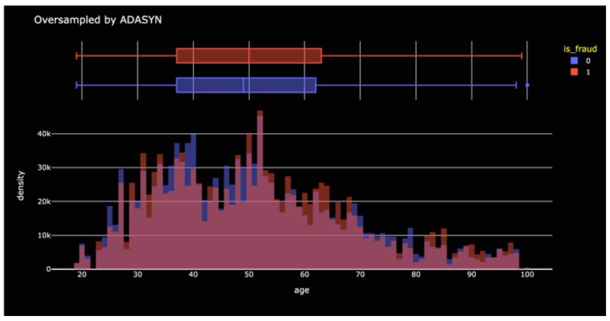


Figure 14 - Oversampled by ADASYN

Under sampling Techniques

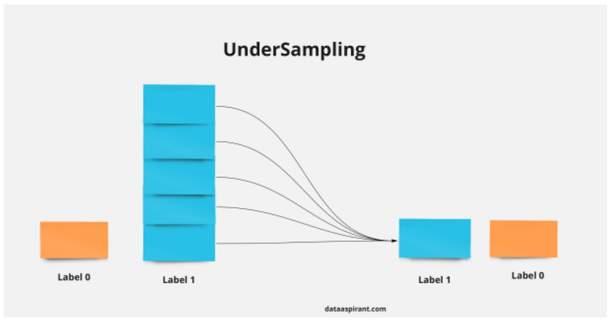


Figure 15 – UnderSampling

Undersampling in machine learning is a technique to address class imbalance by reducing the size of the majority class. It involves randomly removing samples from the overrepresented class to balance the dataset. This is necessary when the minority class is significantly smaller and the model might develop a bias towards the majority class, leading to poor performance on the minority class. Undersampling helps to prevent this by equalizing the class distribution, ensuring the model learns features from both classes effectively. It's particularly useful for large datasets, as it also reduces computational load and improves training efficiency.

RandomUnderSampler

RandomUnderSampler is a technique in machine learning for addressing class imbalance by randomly removing samples from the majority class. It helps create a balanced dataset by equalizing the number of instances in each class, thus ensuring that the model does not become biased towards the majority class.

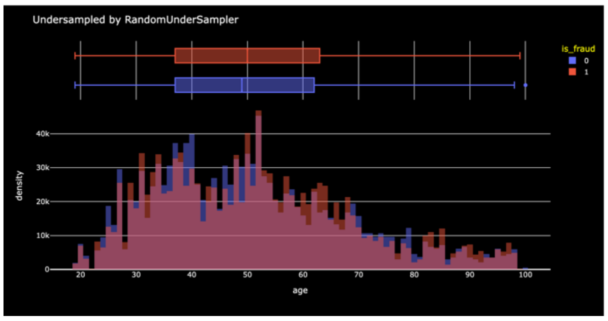


Figure 16 – RandomUnderSampler

Near Miss

NearMiss is an undersampling technique in machine learning that selects samples from the majority class based on their distance to the minority class. It works by keeping only those majority class samples that are nearest to the minority class, thereby ensuring a more representative and balanced dataset for model training.

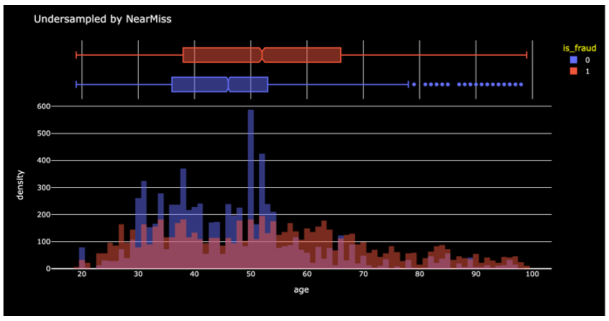
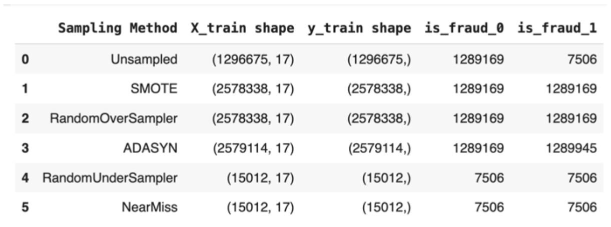


Figure 17 - UnderSampling by Near Miss

Selecting the Sampling Technique

SMOTE (Synthetic Minority Over-sampling Technique) is often preferred in credit card fraud detection for managing class imbalance due to its ability to generate synthetic data, reducing the risk of overfitting compared to RandomOverSampler.



	Sampling Method	X_train shape	y_train shape	is_fraud_0	is_fraud_1
0	Unsampled	(1296675, 17)	(1296675,)	1289169	7506
1	SMOTE	(2578338, 17)	(2578338,)	1289169	1289169
2	RandomOverSampler	(2578338, 17)	(2578338,)	1289169	1289169
3	ADASYN	(2579114, 17)	(2579114,)	1289169	1289945
4	RandomUnderSampler	(15012, 17)	(15012,)	7506	7506
5	NearMiss	(15012, 17)	(15012,)	7506	7506

Figure 18 - Comparison of Shapes after sampling

Unlike under-sampling methods like RandomUnderSampler and NearMiss, SMOTE avoids loss of valuable information by not discarding majority class data. It creates synthetic samples in feature space, enhancing model generalization and providing versatility across various classifiers. While similar to ADASYN in creating synthetic samples, SMOTE's approach is generally more applicable.

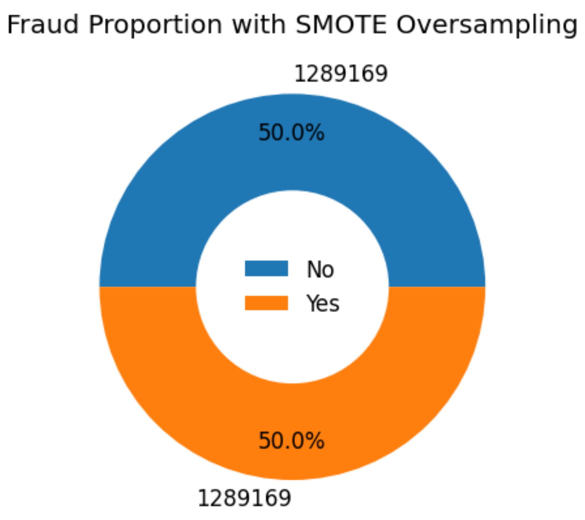


Figure 19 - Fraud proportion after SMOTE

Feature Importance

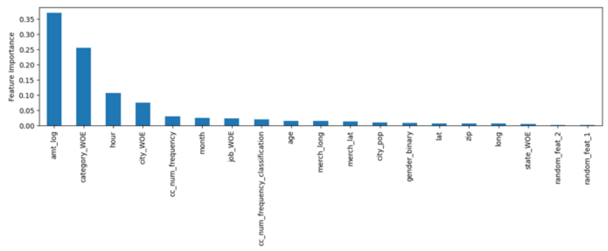


Figure 20 - Feature Importance of features

We used Random Forests to calculate the feature importance are shown in Figure 20.

Calculating feature importance with a Random Forest involves aggregating the importance scores from each decision tree within the ensemble. Each tree in a Random Forest is constructed using a random subset of features and data points. During the construction, the algorithm measures how much each feature decreases the impurity in a split (e.g., using Gini impurity or entropy for classification, mean squared error for regression). These measures are averaged over all trees to determine the overall importance of each feature. The result is a ranking of features based on how much they contribute to the model's predictive power.

From Figure 20, it can be seen that the 4 main features are: amt_log, category_WOE, hour, city_WOE.

Machine Learning Models

Random Forests

It is an ensemble learning method that combines multiple decision trees to improve prediction accuracy and control overfitting. It uses bagging and feature randomness in constructing individual trees, leading to a robust, versatile model.

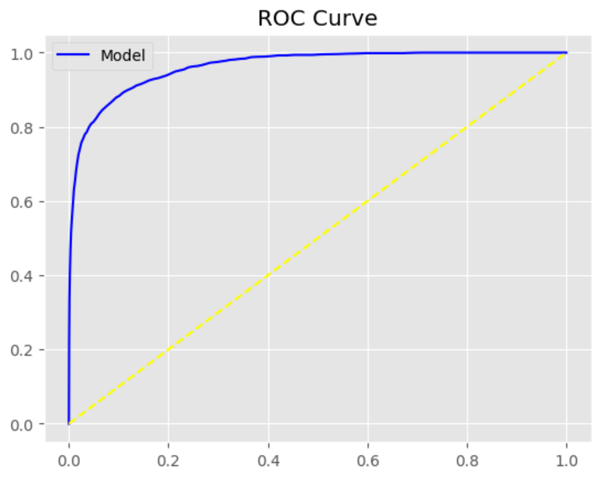


Figure 21 - ROC for Random Forests

K Nearest Neighbours

It is a simple, non-parametric algorithm that predicts a data point's classification based on the most frequent class among its 'k' nearest neighbors. It's intuitive and works well for basic classification tasks.

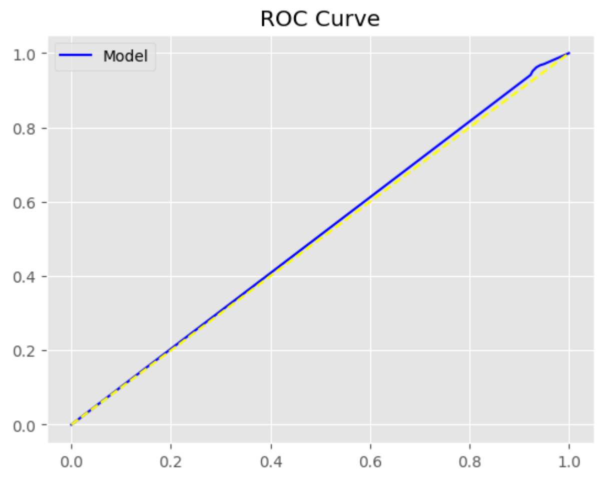


Figure 22 - ROC for K Nearest Neighbours

Gradient Boosting

It is a powerful ensemble technique that builds sequential models, typically decision trees, each correcting the errors of its predecessor, leading to strong predictive performance, especially in structured data scenarios.

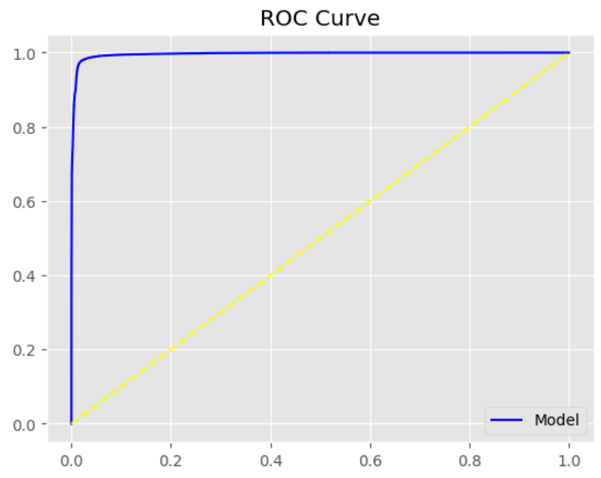


Figure 23 - ROC for Gradient Boosting

Light GBM

A fast, distributed, high-performance gradient boosting framework based on decision tree algorithms, used for ranking, classification, and many other machine learning tasks. It's known for handling large datasets efficiently.

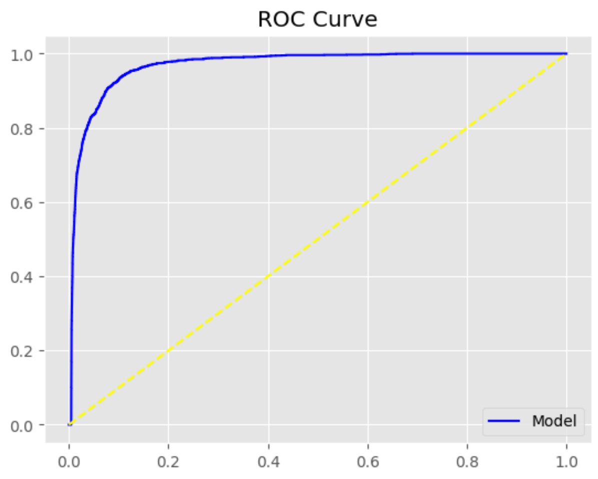


Figure 24 - ROC for Light GBM

Deep Learning Models

Simple Neural Network

It is the foundational neural network structure typically comprising input, hidden, and output layers. It uses interconnected nodes or neurons to process data, suitable for basic tasks like classification and regression.

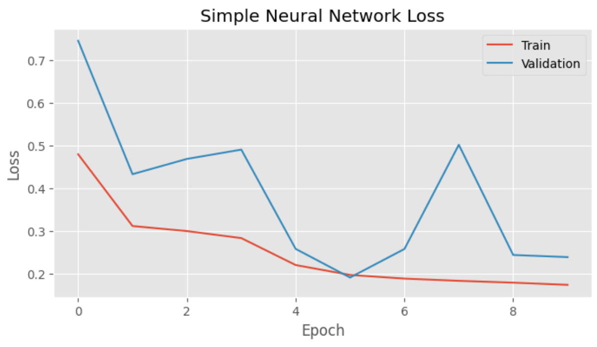


Figure 25 - Loss in Simple Neural Network

Convolutional Neural Network (CNN)

It is a deep learning algorithm specializing in processing structured grid data like images. It employs convolutional layers to automatically and adaptively learn spatial hierarchies of features, ideal for image recognition and processing tasks.

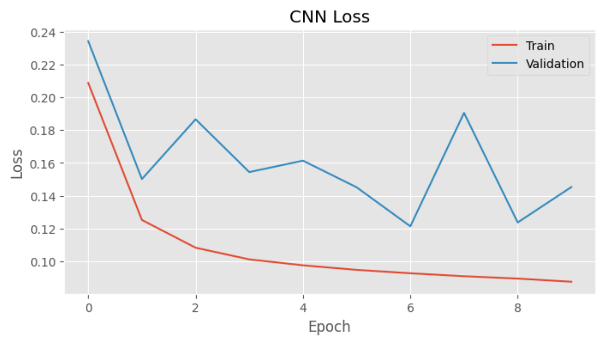


Figure 26 - Loss in CNN

Recurrent Neural Network (RNN)

It is a type of neural network designed for sequential data processing, like time series or language. RNNs have connections that form directed cycles, allowing them to use information from previous steps, making them effective for tasks like language modeling and speech recognition.

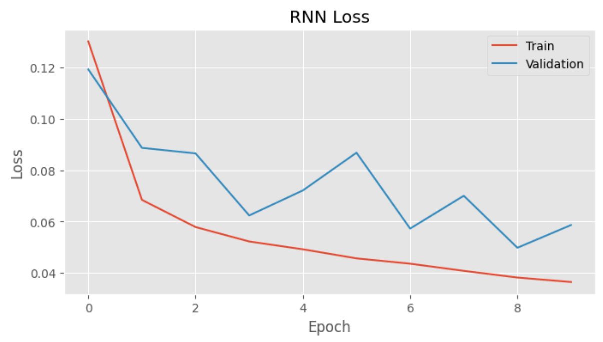


Figure 27 - Loss in RNN

Comparison of all models

Accuracy

Accuracy is a metric that measures the proportion of correct predictions made by a model out of all predictions. It's calculated using the formula:

Precision

Precision is a metric that evaluates the proportion of true positive predictions in the total predicted positives.

Recall

Recall (also known as sensitivity) measures the proportion of actual positives correctly identified by the model. It's crucial when missing a positive case is particularly harmful.

F1 Score

F1 score is a harmonic mean of precision and recall, providing a balance between them. It's particularly useful when the cost of false positives and false negatives is similar. This score is a single metric that combines both precision and recall.

AUC

AUC (Area Under the Curve) refers to the area under the ROC (Receiver Operating Characteristic) curve. This curve plots the true positive rate (Recall) against the false positive rate at various threshold settings. The AUC value, ranging from 0 to 1, quantifies the overall ability of a model to distinguish between classes. Higher values indicate better performance.

Conclusion

Random Forests: It shows a high recall (0.97) indicating it catches most fraud cases but has a very low precision (0.01), which means it also misclassifies many non-fraud cases as fraud. However, it has a high AUC (0.9628), suggesting good separability between classes.

K Nearest Neighbors (KNN): This model performs poorly across all metrics except recall, with an accuracy of just 0.07. The AUC (0.5096) is barely better than random guessing, indicating that KNN is not a good model for this dataset.

Gradient Boosting: It has perfect recall (1.0), meaning it identifies all fraud cases but suffers from low precision, similar to Random Forests. The AUC is very high (0.9880), which is encouraging for potential improvements.

Light GBM: Similar to Gradient Boosting, it has perfect recall but low precision. The AUC is high (0.9694), indicating a good model with room for optimization.

Simple Neural Network and CNN: Both have significantly better balance across all metrics, with the Simple Neural Network achieving the highest accuracy (0.92). CNN has a slightly lower performance in comparison but still decent.

RNN: This model has moderate performance metrics across the board, with no particular strengths or weaknesses highlighted in the provided metrics.

In conclusion, while traditional models like Random Forests and Gradient Boosting show high sensitivity to fraud detection, they lack precision and consequently may produce many false positives. The neural network-based models, especially the Simple Neural Network, provide the best overall performance in terms of accuracy and balance between precision and recall. This suggests that deep learning models are more suitable for this task, potentially due to their ability to capture complex patterns in the data. However, the precision of all models is generally low, indicating a need for further tuning and possibly more data to improve the distinction between fraudulent and non-fraudulent transactions.

Model Name	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**AUC**
Random Forests	0.72	0.01	0.97	0.03	**0.9628**
K Nearest Neighnours	0.07	0.0	0.96	0.01	0.5096
Gradient Boosting	0.53	0.01	1.0	0.02	0.9880
Light GBM	0.33	0.01	1.0	0.01	0.9694
Simple Neural Network	**0.92**	0.52	0.93	0.52	
CNN	0.84	0.51	0.91	0.48	
RNN	0.63	0.51	0.81	0.40	

Table 1 - Model Evaluation Metrics