**Credit Card Fraud Detection using Random Forest**

**Introduction**

Credit card fraud poses a significant threat to financial institutions and consumers worldwide. Detecting fraudulent transactions in a timely and accurate manner is crucial for minimizing losses and maintaining trust in the banking system. In this report, we explore the application of Random Forest, a powerful machine learning algorithm, for the task of credit card fraud detection.

**Why Random Forest?**

Random Forest is a versatile ensemble learning method known for its robustness and effectiveness in handling complex classification tasks. By combining multiple decision trees and leveraging the concept of bagging, Random Forest can effectively identify patterns and anomalies in large and high-dimensional datasets, making it well-suited for credit card fraud detection.

**Importing the Libraries**

numpy and pandas

These libraries are fundamental for data manipulation and analysis. They provide essential tools for handling datasets, performing mathematical operations, and organizing data structures efficiently.

matplotlib and seaborn

These visualization libraries enable the creation of insightful plots and graphs, facilitating exploratory data analysis and the visualization of key insights. Matplotlib provides a flexible interface for creating a wide range of plots, while Seaborn offers enhanced styling and additional statistical functionality.

matplotlib.gridspec

The gridspec module from Matplotlib allows for more advanced layout customization of subplots within a figure. This functionality is useful for creating complex multi-panel plots with precise control over the arrangement of individual subplots.

**Exploratory Data Analysis**

**Determining Number of Fraud Cases in Dataset**

The code snippet below calculates the proportion of fraud cases relative to valid transactions in the dataset. This analysis provides valuable insights into the imbalance between fraudulent and non-fraudulent transactions, which is essential for understanding the nature of the problem and selecting appropriate modeling techniques.

fraud = data[data['Class'] == 1]

valid = data[data['Class'] == 0]

outlierFraction = len(fraud) / float(len(valid))

print(outlierFraction)

print('Fraud Cases: {}'.format(len(data[data['Class'] == 1])))

print('Valid Transactions: {}'.format(len(data[data['Class'] == 0])))

By quantifying the ratio of fraud cases to valid transactions and printing the total number of each, this analysis provides a clear picture of the dataset's class distribution, which is crucial for designing and evaluating the performance of the fraud detection model.

**Correlation Matrix**

**Purpose**

The correlation matrix is generated to examine the relationships between different features in the dataset. This analysis helps identify potential correlations between variables, which can provide insights into the underlying patterns and dependencies within the data.

**Code Explanation**

corrmat = data.corr()

fig = plt.figure(figsize=(12, 9))

sns.heatmap(corrmat, vmax=.8, square=True)

plt.show()

The code calculates the correlation coefficients between all pairs of features in the dataset using the .corr() function. The resulting correlation matrix (corrmat) is then visualized as a heatmap using Seaborn's heatmap() function. The heatmap color intensity represents the strength and direction of the correlation, with higher values indicating stronger correlations.

**Visualization**

The heatmap provides a visual representation of the correlation matrix, where each cell corresponds to the correlation coefficient between two features. The color scale ranging from light to dark indicates the strength of correlation, with lighter colors indicating stronger positive correlations and darker colors indicating stronger negative correlations. This visualization allows for quick identification of potential patterns and relationships between variables, aiding in feature selection and model interpretation.

**Declare Feature Vector and Target Variable**

Purpose

This step involves defining the feature vector (X) and target variable (y) required for training the Random Forest Classifier. The feature vector contains the independent variables, while the target variable represents the outcome or class labels to be predicted.

**Split Data into Separate Training and Test Set**

Purpose

Splitting the dataset into separate training and test sets is essential for evaluating the performance of the Random Forest Classifier. The training set is used to train the model, while the test set is reserved for evaluating its performance on unseen data.

**Using Random Forest Classifier**

**Building the Random Forest Classifier (RANDOM FOREST)**

Importing Random Forest Classifier: The Random Forest Classifier is imported from the sklearn.ensemble module. This classifier is known for its ability to handle complex classification tasks by leveraging ensemble learning techniques.

Random Forest Model Creation: An instance of the Random Forest Classifier (rfc) is created without specifying any hyperparameters. The classifier is then trained on the training data (xTrain and yTrain) using the fit() method.

Making Predictions: The trained Random Forest Classifier (rfc) is used to make predictions on the test data (xTest). Predictions are stored in the variable yPred.

**Model Evaluation**

Model Accuracy: The accuracy of the Random Forest Classifier on the test set is calculated using the score() method, which compares the predicted labels (yPred) with the actual labels (yTest).

Misclassified Samples: The number of misclassified samples is calculated and printed to provide insights into the classifier's performance.

**Feature Importance Analysis**

Feature Importance Calculation: The feature importance scores are calculated using the feature\_importances\_ attribute of the trained Random Forest Classifier (rfc). These scores represent the contribution of each feature to the predictive performance of the model.

**Visualizing Important Features:** A bar plot is created to visualize the important features, with feature importance scores on the x-axis and feature names on the y-axis. This visualization helps identify the most influential features in the dataset, aiding in feature selection and model interpretation.

**Evaluation Metrics**

**Evaluating the Classifier**

The evaluation of the Random Forest Classifier involves assessing its performance using various classification metrics. These metrics provide insights into different aspects of the classifier's effectiveness in correctly identifying fraudulent transactions and distinguishing them from valid ones.

**Why Individual Metrics are Used:**

**Accuracy:** Accuracy measures the proportion of correctly classified instances out of the total instances. It provides a general overview of the classifier's performance but may not be suitable for imbalanced datasets where the classes are unevenly distributed.

**Precision:** Precision measures the proportion of true positive predictions out of all positive predictions made by the classifier. It indicates the classifier's ability to avoid misclassifying valid transactions as fraudulent, minimizing false positives.

**Recall:** Recall, also known as sensitivity or true positive rate, measures the proportion of true positive predictions out of all actual positive instances in the dataset. It evaluates the classifier's ability to correctly identify fraudulent transactions, minimizing false negatives.

**F1-Score:** The F1-Score is the harmonic mean of precision and recall. It provides a balance between precision and recall and is particularly useful when there is an uneven class distribution or when both false positives and false negatives are equally important.

**Matthews Correlation Coefficient (MCC):** MCC is a correlation coefficient between the observed and predicted binary classifications. It takes into account true and false positives and negatives and is particularly useful for imbalanced datasets, providing a balanced measure of classifier performance.

**Printing the Confusion Matrix**

Purpose

The confusion matrix provides a detailed breakdown of the classifier's predictions and the actual class labels in a tabular format. This analysis helps assess the classifier's performance in correctly identifying normal and fraudulent transactions, as well as understanding the types of errors made.

Code Explanation

The provided code generates a heatmap visualization of the confusion matrix using Seaborn's heatmap() function. The matrix displays the counts of true positive, true negative, false positive, and false negative predictions, allowing for a comprehensive assessment of the classifier's performance.

**Extracting Additional Metrics from Confusion Matrix**

After visualizing the confusion matrix, additional metrics are extracted and calculated:

**Percentage of Fraud Cases:** The percentage of fraud cases from the total is calculated to provide insights into the prevalence of fraudulent transactions in the dataset.

**True Positive, True Negative, False Positive, False Negative:** These values represent the counts of each type of prediction outcome, further dissecting the classifier's performance.

**Calculating Specificity, False Positive Rate, and False Negative Rate**

Specificity, false positive rate (FPR), and false negative rate (FNR) are calculated to provide additional insights into the classifier's performance, particularly in terms of its ability to correctly identify normal transactions and minimize misclassifications.

**Calculating Area Under the ROC Curve (AUC-ROC)**

The area under the Receiver Operating Characteristic (ROC) curve (AUC-ROC) is calculated to evaluate the classifier's ability to distinguish between classes. A higher AUC-ROC value indicates better discrimination between normal and fraudulent transactions.

**Naive Bayes Classifier for Credit Card Fraud Detection**

In addition to using the Random Forest Classifier, the Naive Bayes Classifier is employed to model the credit card fraud detection dataset. Naive Bayes is a probabilistic classifier based on Bayes' theorem with the "naive" assumption of independence between features. Despite its simplicity, Naive Bayes can often perform well on text classification and other tasks with high-dimensional data.

**Why Naive Bayes Classifier?**

The Naive Bayes Classifier is chosen as an alternative to the Random Forest Classifier due to its simplicity, efficiency, and effectiveness in handling high-dimensional data. Additionally, Naive Bayes is well-suited for binary classification tasks and can provide insights into how different modeling approaches perform on the same dataset.

**Modeling with Naive Bayes Classifier**

Code Explanation

The Naive Bayes Classifier is trained and evaluated using similar steps as the Random Forest Classifier. The classifier is imported from the appropriate module (sklearn.naive\_bayes) and then trained on the training data (xTrain and yTrain). Predictions are made on the test data (xTest), and evaluation metrics such as accuracy, precision, recall, F1-score, Matthews correlation coefficient, and confusion matrix are calculated and compared to those obtained from the Random Forest Classifier.

**Comparison Using Evaluation Metrics**

Both the Random Forest and Naive Bayes classifiers are evaluated using the same set of evaluation metrics to facilitate a fair comparison. These metrics provide insights into the classifiers' performance in correctly identifying fraudulent transactions and distinguishing them from valid ones.

**Conclusion**

In conclusion, after thorough evaluation of both the Random Forest and Naive Bayes classifiers for credit card fraud detection, several key findings emerged.

* While the accuracy remains consistent between both models, the precision of the Naive Bayes Classifier lags significantly behind the Random Forest Classifier.
* The percentage of fraud cases detected is slightly lower in the Naive Bayes Classifier compared to the Random Forest Classifier.
* The Random Forest Classifier demonstrates slightly superior performance in terms of AUC-ROC, recall, and Matthews correlation coefficient.
* F1 score and specificity exhibit comparable results across both classifiers.
* Notably, the False Negative Rate is lower in the Random Forest Classifier, indicating its superior ability to correctly identify fraudulent transactions without misclassifying them as valid.

Considering all these metrics collectively, it is evident that the Random Forest Classifier outperforms the Naive Bayes Classifier for credit card fraud detection. Therefore, I have decided to utilize the Random Forest Classifier for this task, owing to its superior performance across multiple evaluation criteria and its ability to effectively identify and mitigate fraudulent transactions.