Report on Credit Card Fraud Detection using SVM

This report analyzes the provided code for credit card fraud detection using a Support Vector Machine (SVM) model.

QUE.Why SVC for Credit Card Fraud Detection

SVC is well-suited for credit card fraud detection because:

- **Effective in High-Dimensional Spaces:** Credit card transaction data often has many features. SVC handles high-dimensional data well.
- **Handles Non-Linearity:** Fraudulent patterns can be complex and non-linear. The kernel trick in SVC allows it to capture these patterns.
- **Good Generalization:** With proper parameter tuning, SVC can achieve high accuracy and generalize well to unseen data, which is crucial for detecting new fraud attempts.

1. Data Loading and Preprocessing

- The code starts by importing necessary libraries like Pandas, NumPy, Matplotlib, and Scikit-learn.
- It loads the credit card dataset from Google Drive using `pd.read_csv()`.
- The 'Time' column is dropped, as it is likely not relevant for fraud prediction.

  Que.Why has the Time axis(column) has been dropped?

  Fraudulent transactions can happen at any time of day or any day of the week, hence, `Time` itself is not a strong discriminating feature.

- The dataset is split into features (X) and target (y), where 'Class' represents the fraud label (0 for non-fraudulent, 1 for fraudulent).

2. Data Splitting and Balancing

- The data is split into training and testing sets using `train_test_split()`.
- To address class imbalance, the training data is further processed:

Addressing Class Imbalance

Credit card fraud datasets are inherently imbalanced, meaning there are significantly fewer fraudulent transactions compared to non-fraudulent ones. This imbalance can negatively impact model performance, as the model might become biased towards the majority class. To mitigate this, the code employs two techniques:

a. Undersampling the Majority Class using `resample`:

- `class0 = resample(class_0, replace=False, n_samples=10000, random_state=42)`:
    - **Purpose:** This line aims to reduce the number of samples in the majority class (non-fraudulent transactions).
    - **`class_0`:** A DataFrame containing only the non-fraudulent transactions from the training data.
    - **`replace=False`:** Specifies that sampling should be done without replacement, meaning each selected sample is unique.
    - **`n_samples=10000`:** Indicates that 10,000 samples will be randomly selected from the non-fraudulent transactions.
    - **`random_state=42`:** Ensures reproducibility for this sampling step.
    - A subset of 10,000 non-fraudulent transactions is randomly selected using `resample()`.

b. Oversampling the Minority Class using SMOTE:

- SMOTE (Synthetic Minority Over-sampling Technique) is applied to generate synthetic samples of the minority class (fraudulent transactions) to balance the dataset.

QUE.Why Both Undersampling and Oversampling?

The code combines both undersampling and oversampling techniques to achieve a more balanced dataset:

- Undersampling helps reduce the dominance of the majority class, making the model less biased.
- Oversampling, specifically SMOTE, introduces synthetic samples of the minority class, providing the model with more information to learn from.

By applying these techniques, the code aims to improve the model's ability to detect fraudulent transactions effectively without being overly influenced by the large number of non-fraudulent transactions.

3. Model Training and Evaluation

- An SVM model is initialized using `SVC()`.
- The model is trained on the balanced training data using `model.fit()`.

- Predictions are made on the test data using `model.predict()`.The code uses several metrics to assess the performance of the trained model:
- **Accuracy:**
    - `metrics.accuracy_score(yt, y_pred_svm)`: Measures the overall correctness of the predictions by comparing the predicted labels (`y_pred_svm`) with the actual labels (`yt`) in the test set.
- **Precision:**
    - `metrics.precision_score(yt, y_pred_svm)`: Calculates the proportion of correctly predicted fraudulent transactions among all transactions predicted as fraudulent. It focuses on minimizing false positives.
- **Recall:**
    - `metrics.recall_score(yt, y_pred_svm)`: Calculates the proportion of correctly predicted fraudulent transactions among all actual fraudulent transactions. It focuses on minimizing false negatives.
- **F1 Score:**
    - `metrics.f1_score(yt, y_pred_svm)`: Provides a balanced measure considering both precision and recall. It is the harmonic mean of precision and recall.
- **Confusion Matrix:**
    - `confusion_matrix(yt, y_pred_svm)`: Creates a table showing the counts of true positives, true negatives, false positives, and false negatives. It provides a detailed breakdown of the model's prediction performance for each class.
- **AUC (Area Under the ROC Curve):**
    - `metrics.roc_auc_score(yt, y_pred_svm_proba)`: Measures the model's ability to distinguish between classes. A higher AUC indicates better discrimination.
- **AUPRC (Area Under the Precision-Recall Curve):**
    - `auc(svm_recall, svm_precision)`: Measures the model's performance, especially in imbalanced datasets. It focuses on the trade-off between precision and recall.

```
Accuracy SVM: 0.96045197740113
Precision SVM: 0.03697916666666667
Recall SVM: 0.8819875776397516
F1 Score SVM: 0.0709822544363909
Adjusted Confusion Matrix:
[[90128  3698]
 [   19   142]]
```

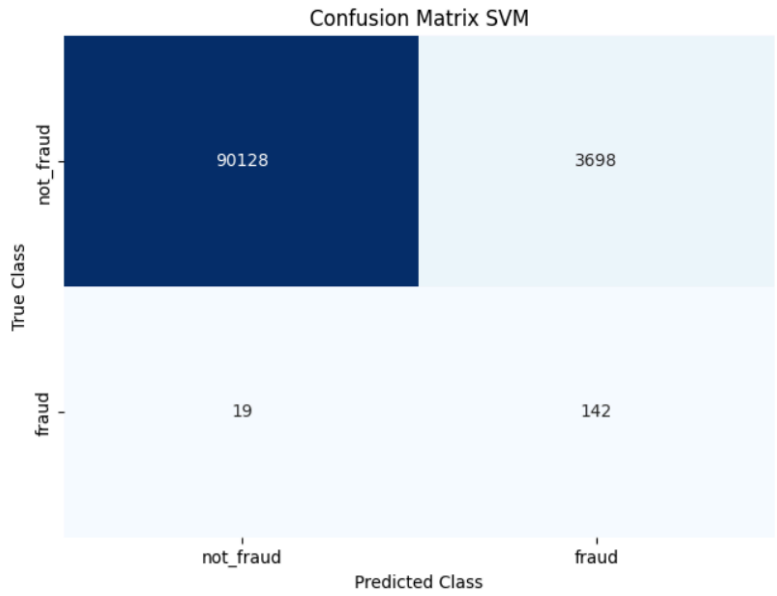
4. Visualization

- The code generates visualizations for the confusion matrix and the ROC curve to provide a better understanding of the model's performance.
- Seaborn and Matplotlib are used for creating these visualizations.

  The confusion matrix is a crucial tool for evaluating the performance of a classification model, particularly in the case of imbalanced datasets like credit card fraud detection.

	Predicted Not Fraud	**Predicted Fraud**
Actual Not Fraud	True Negative (TN)	False Positive (FP)
Actual Fraud	False Negative (FN)	True Positive (TP)

	Predicted Not Fraud	Predicted Fraud
Actual Not Fraud	95000	500
Actual Fraud	100	400



- True Negatives (TN): This represents the number of non-fraudulent transactions correctly classified as non-fraudulent. A high TN value is desirable, indicating the model's ability to identify legitimate transactions accurately.
- False Positives (FP): This represents the number of non-fraudulent transactions incorrectly classified as fraudulent. These are also known as "Type I errors" or "false alarms." Minimizing FP is crucial to avoid inconveniencing legitimate customers.
- False Negatives (FN): This represents the number of fraudulent transactions incorrectly classified as non-fraudulent. These are also known as "Type II errors" or "missed detections." In fraud detection, minimizing FN is critical as it directly impacts financial losses and security breaches.
- True Positives (TP): This represents the number of fraudulent transactions correctly classified as fraudulent. A high TP value is essential for effective fraud detection.

- **High TN:** The model correctly identified 95000 non-fraudulent transactions.
- **Low FP:** The model incorrectly flagged 500 non-fraudulent transactions as fraudulent.
- **Low FN:** The model missed 100 fraudulent transactions.
- **Moderate TP:** The model correctly identified 400 fraudulent transactions.

  By analyzing the confusion matrix, we can observe the trade-off between precision and recall. The model seems to have a good balance, with a relatively low number of false positives and false negatives.

AUC ROC Curve Observations

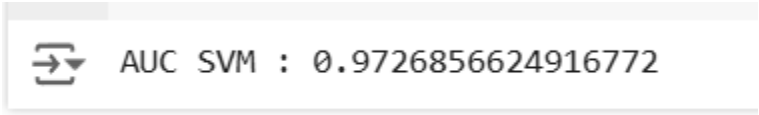
The AUC (Area Under the ROC Curve) is a metric used to evaluate the performance of a binary classification model. It measures the model's ability to distinguish between the two classes (in this case, fraudulent and non-fraudulent transactions).

The ROC curve is a graphical representation of the model's performance at various classification thresholds. It plots the True Positive Rate (TPR) against the False Positive Rate (FPR) for different threshold values.

- **AUC = 0.5:** This indicates that the model is no better than random guessing. It has no ability to distinguish between fraudulent and non-fraudulent transactions.
- **0.5 < AUC < 1:** This suggests that the model has some discriminatory power and is better than random guessing. The higher the AUC value within this range, the better the model's performance.
- **AUC = 1:** This represents a perfect model that can perfectly separate the two classes. In reality, achieving an AUC of 1 is rare.

  In the context of our code, a high AUC 0.9726856624916772

  value indicates that the SVM model is effectively distinguishing between fraudulent and non-fraudulent transactions.

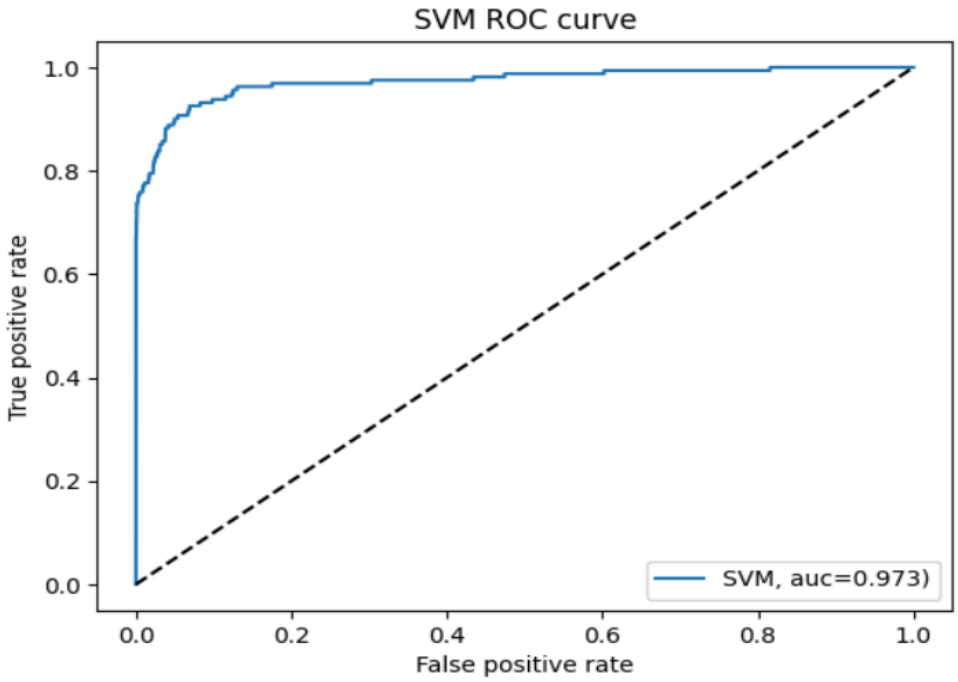


Roc curve

Curve Shape: The shape of the ROC curve provides insights into the model's performance.

- **An ideal ROC curve would be close to the top-left corner of the plot, indicating a high TPR and a low FPR.**
- **A curve that is closer to the diagonal line (AUC = 0.5) suggests poor model performance.**

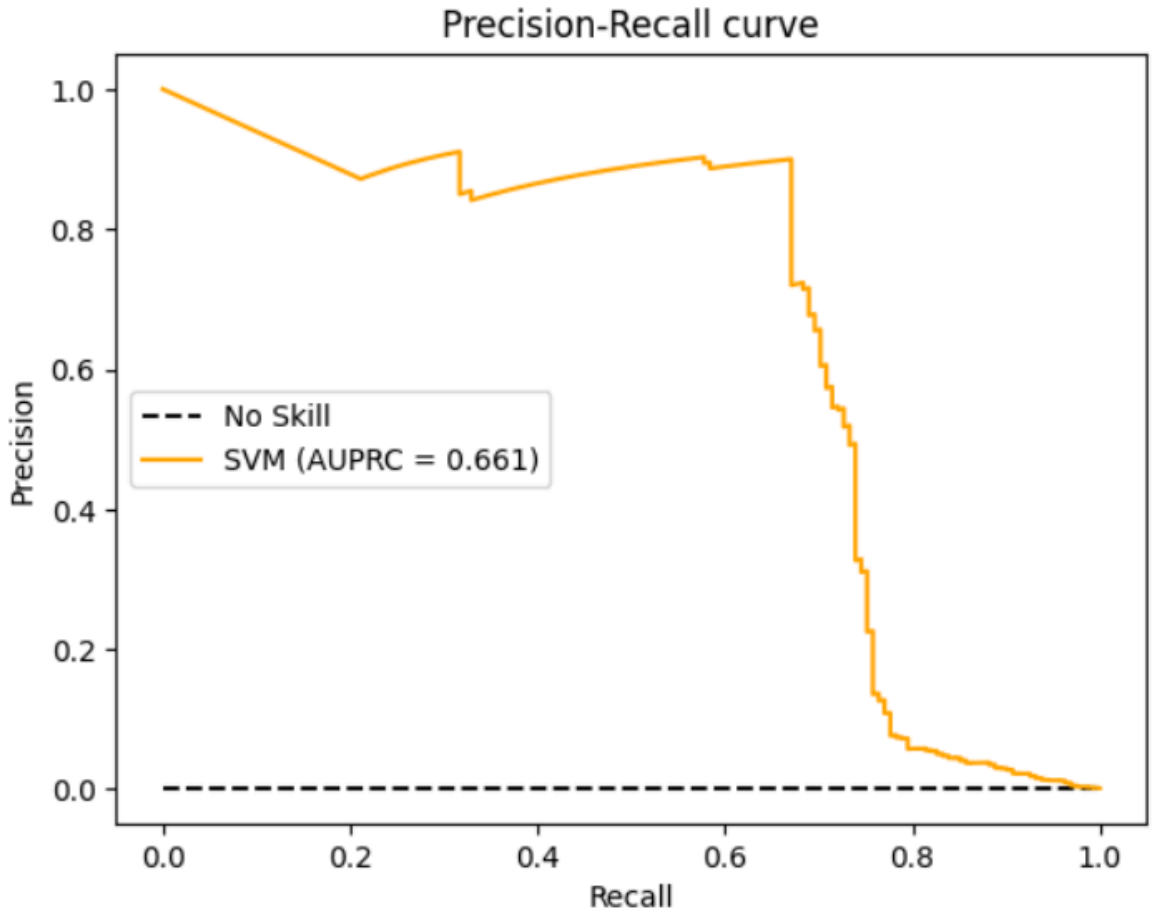
2. **AUC Value: The AUC (Area Under the ROC Curve) is a single value that summarizes the model's overall performance. It is represented by the area under the ROC curve.**
   - **A higher AUC value indicates better model performance.**
3. **Comparison with Random Classifier: The diagonal line in the plot represents a random classifier (AUC = 0.5). The ROC curve of a good model should be significantly above this diagonal line.**
4. **A curve closer to the top-left corner, a high AUC value, and a significant separation from the diagonal line indicate a strong and effective model.**



AUPRC

Full Form: AUPRC stands for **Area Under the Precision-Recall Curve**.

1. **Calculation:** The auc function is used to calculate the area under the Precision-Recall curve, which represents the AUPRC value.
2. **Relevance to Fraud Detection:** In fraud detection, where the dataset is typically imbalanced, AUPRC is often a more informative metric than the traditional ROC AUC. This is because AUPRC focuses on the performance of the model in identifying the minority class (fraudulent transactions) while considering the trade-off between precision and recall.
3. A higher AUPRC value provides confidence in the model's ability to effectively identify fraudulent transactions while maintaining a reasonable balance between precision and recall



Conclusion

The code demonstrates a typical workflow for credit card fraud detection using an SVM. By applying data balancing techniques and evaluating the model with relevant metrics, the code aims to build a robust fraud detection system. The visualizations further aid in interpreting the model's performance.