Machine Learning for Finance Fraud Detection

Pranav Prabhu
Virginia Tech
Virginia Polytechnic Institute and State University, Blacksburg, VA 24061-0002
ppranav02@vt.edu

Abstract

In an era of digital finance, the increased volume of digital transactions has led to many fraudulent activities taking place around the world. With the advent of modern technology and continued economic growth, acts of fraud have become increasingly prevalent, costing institutions billions of dollars annually. Fraudsters are continuously evolving their approaches to exploit the vulnerabilities of the current prevention measures in place, many of whom are targeting the financial sector (Hilal et al). These crimes continue to take place regularly in numerous industries and sectors such as automobile and healthcare insurance fraud, credit card fraud, etc. This scenario highlights the importance of developing new innovative approaches to combat fraudulent activities while safeguarding the integrity of financial transactions.

Machine Learning (ML) offers a promising solution to this problem by leveraging advanced algorithmic models that can learn from, adapt to, and detect fraudulent activities autonomously. These ML models do so by analyzing large datasets of transactional data, financial records, customer behavior, and numerous other features to identify patterns and anomalies in financial data, which can significantly help combat finance crimes. This project aims to leverage various predictive machine learning models that train on big datasets to identify and predict fraudulent financial transactions.

1. Introduction

In my project, I set out to tackle the problem of identifying fraudulent credit card transactions. The objective was to create a system predict and identify fraudulent transactions as accurately as possible. Currently, this detection is often done by a mix of computer programs and human oversight. However, the systems in place can be slow and aren't always good at catching fraud, especially as methods of committing fraud are constantly changing. These systems can also make mistakes, like flagging normal transactions as fraud, which can be frustrating for credit card users. Abdulalem Ali, Waleed Hilal, and Rejwan Bin Sulaiman suggest the use of various ML approaches, emphasizing the importance of classifiers such as KNNs, SVMs, and ANNs; however some of these classifiers have limited performance in that they can take exponential time to learn model hyper-parameters and may not widely suitable for the task. The impact of my work is important for anyone who uses a credit card. If I succeed, it would mean fewer fraudulent transactions slipping through the cracks, saving money for both the cardholders and the banks. It would also mean fewer mistakes in flagging normal transactions as fraudulent, making using credit cards smoother and more trustworthy for everyone.

2. Approach

This project employed a multi-step approach to detect anomalies and predict fraudulent transactions. The approach included several key steps as listed below:

A. **Data Pre-processing:** Initial steps included handling the dataset by scaling features that weren't transformed by PCA, namely 'Time' and 'Amount'. This standardization was necessary to avoid any bias due to the varying scales of data.

B. **Handling Imbalanced Data:** The dataset includes credit card transactions performed in September 2013 by European cardholders. This dataset includes transactions that occurred in two days, with 492 frauds (0.172%) out of 284,807 transactions. Given the discrepancy between fraudulent and non-fraudulent transactions, methods such as random undersampling and oversampling with SMOTE (Synthetic Minority Over-sampling Technique) were used to keep the models from favoring the majority (non-fraudulent) class.

C. **Model Training and Validation:** Various predictive models including Logistic Regression, K-Nearest Neighbors, Support Vector Machines, and Decision Trees were trained. The training was conducted on both under-sampled and over-sampled datasets to compare their effectiveness.

D. **Hyper-parameter Tuning:** Methods like RandomizedSearchCV and GridSearchCV were utilized to fine-tune the models and improve their predictive accuracy.

E. **Evaluation Metrics:** The models were evaluated based on accuracy, precision, recall, F1 score, and the Area Under the ROC Curve (AUC-ROC), which are critical in assessing the performance of imbalanced classes.

I expected a few hiccups along the way as I was not fully familiar with all of the Python libraries and ML tools/frameworks that could be used for training my models. And indeed not everything worked perfectly from the get-go. The primary challenge was the skewed distribution of the classes. Given the complexity of the models and the nature of the data, there was a concern about models overfitting the training data. Initially, the models showed poor performance when tested on validation sets due to the imbalanced nature of the dataset. The initial attempts involved the straightforward application of the classification models on the imbalanced dataset, which, did not yield satisfactory results due to high class imbalance.

3. Experiments and Results

Success was measured using a combination of the following key metrics:

A. **Accuracy:** While not the primary metric due to the data imbalance, it provided a basic understanding of overall model performance.

B. **Precision and Recall:** These metrics were crucial to balance the trade-offs between false positives and false negatives. High precision ensures that non-fraudulent transactions are not misclassified as fraudulent, while high recall is important to ensure that fraudulent transactions are not missed.

C. **F1-Score:** F1 score is calculated as the average of precision and recall, with the relative contribution of both measures equal to F1 score. The best F1 score is 1, while the worst is 0.

D. **Area Under the ROC Curve:** Diagnostic test accuracy was measured using the AUC-ROC curve. The test's accuracy increases as the ROC curve approaches the upper left corner of the graph, when the sensitivity is 1 and the false positive rate is 0 (specificity is 1). Thus, the ideal ROC curve has an AUC score of 1.0.

Starting from the very first section, we can see a visualization of the histograms of the distribution of transaction time and amount and the skewed nature of the "Amount" graph with majority of transactions being very small (mean value of $88).

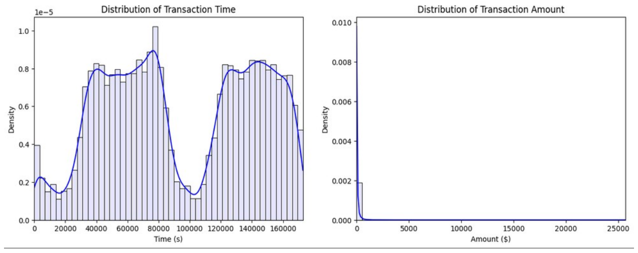

Figure 1: histograms for the distribution of the 'Time' and 'Amount' features, overlaid with kernel density estimation (KDE) curves.

Once we split the data, in the random under-sampling section, we create a count-plot to plot the distribution of the classes in the sub-sample dataset.

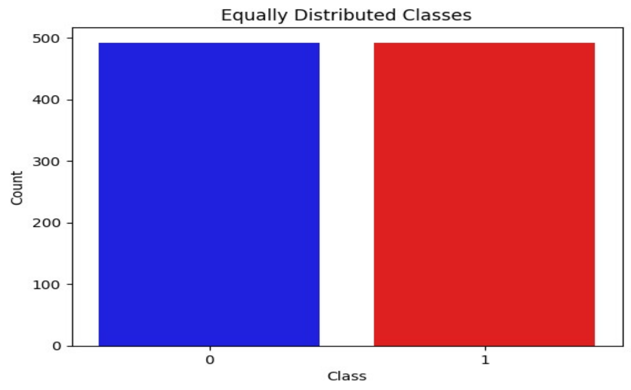

Figure 2: count-plot of distribution of fraudulent and non-fraudulent classes in the randomly under-sampled dataset.

The AUC-ROC curve for all 4 major classifiers are also plotted as shown below with Logistic Regression dominating the other three classifiers.

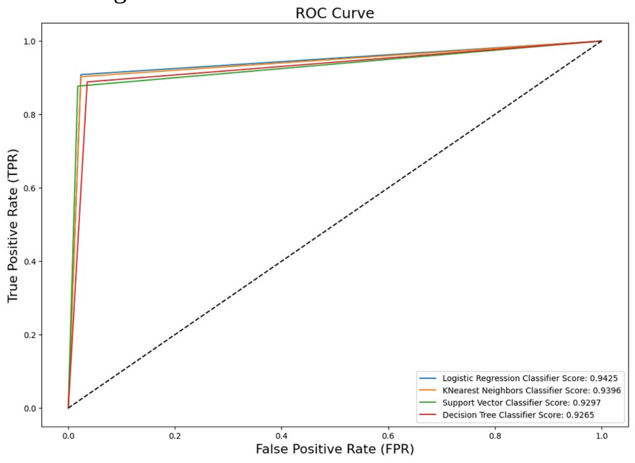

Figure 3: AUC-ROC curves and their scores for all 4 major classifiers – logistic regression, KNN, SVM, and decision trees

Results:

Models trained on the balanced dataset (both undersampled and oversampled) showed improved performance metrics. For example, logistic regression consistently exhibited better performance in terms of accuracy, precision, recall, F1-scores, and average precision-recall scores with even better performance on SMOTE oversampled data compared to undersampled data. The AUC-ROC scores were above 0.90 for all classifiers, indicating a strong ability to distinguish between fraudulent and non-fraudulent transactions.

The use of SMOTE generally led to better model sensitivity (higher recall), which were crucial to identify fraudulent transactions. Hyper-parameter tuning helped in finding the optimal settings that maximized the effectiveness of each model. Overall, the project was successful in significantly improving the ability to detect fraudulent transactions using various ML models; however, they still faced challenges with precision in some cases (notably in SMOTE), where the increase in recall came at the cost of more false positives.

4. Availability

My code is publicly available on GitHub here. The code is released under an MIT License, which is a permissive open-source license. This choice is motivated by the desire to allow others to freely use, modify, and distribute the software. The complete source code and documentation are available on GitHub.

5. Reproducibility

The "Credit Card Fraud Detection" dataset used in this project includes transactions made by credit cards in September 2013 by European cardholders, which is a publicly available dataset from Kaggle. All models used in the project—Logistic Regression, K-Nearest Neighbors, Support Vector Machines, and Decision Trees—are configured with specific parameters. Where applicable, default parameters are used unless otherwise optimized through hyper-parameter tuning. The hyper-parameter tuning process involved using both RandomizedSearchCV and GridSearchCV. The exact parameters explored and the best parameters found are explicitly stated, which helps in replicating the model training process. The code for calculating these metrics is included in the project, ensuring that others can achieve the same results if they follow the same steps, although the exact output varies from run to run.

6. Problem Structure

As mentioned previously, the main objective of this project is to an ML model that could accurately detect fraudulent transactions given a highly unbalanced dataset. The dataset consisted primarily of non-fraudulent transactions, with a very small percentage being fraudulent. This imbalanced posed a significant challenge, as it can lead to developing biased models if not handled properly. To address this issue, the model structure was designed to incorporate the following strategies specifically aimed at handling the imbalance:

A. **Data Resampling:** I experimented with both random under-sampling and SMOTE oversampling techniques to study the performance of the various classifiers on both types of data. These methods helped balance the dataset, ensuring that the models do not become biased towards the majority class. This is reflected in the model training where separate models were trained on under-sampled and oversampled datasets to compare performances.

B. **Multiple Classifiers:** Employing various machine learning models such as Logistic Regression, K-Nearest Neighbors, Support Vector Machines, and Decision Trees allowed for testing which models best handle the imbalanced data and effectively detect fraud. Also, as noted earlier, Logistic Regression proved to be the most rigorous classifier on both under-sampled and over-sampled datasets.

C. **Evaluation Metrics:** Given the problem's nature, standard accuracy alone was not a reliable metric. Hence, precision, recall, F1 score, and AUC-ROC were used to evaluate model performance more comprehensively.

7. Model Parameters

Several aspects of my model had learned parameters as noted below:

A. Logistic Regression: This model uses learned parameters such as weights and biases, which were adjusted during the training process to minimize the loss function – focal loss.

B. K-Nearest Neighbors: While KNN is a non-parametric model that doesn't learn any parameters during training, it relies in the hyper-parameter 'n_neighbors' which defines how many nearest points to consider.

C. Support Vector Machines: SVM has parameters such as kernel type and regularization parameter ('C'), which learns weights and an optional bias.

D. Decision Trees: Trees learn parameters such as splits at each node and the depth of each tree.

Some parts of the models such as scaling using 'StandardScaler', StratifiedShuffleSplit (used for splitting the data into training and testing sets), decision functions and prediction thresholds, cross-validation, and hyper-parameter tuning techniques such as RandomizedSearchCV and GridSearchCV did not have learned parameters but were crucial for post-processing.

8. Model Input/Output

The models expected specific representations of input and output, with several steps taken for data pre-processing and post-processing to ensure the data was suitable for effective model training and evaluation.

A. **Input Representation:** The original dataset contained features transformed through Principal Component Analysis (PCA) resulting in principal components V1 to V28 (hidden for privacy), which were the main input features for the models. The scaled features – Time and Amount - were added back into the dataset to replace the original "Time" and "Amount" features for model training.

B. **Output Representation:** The output for the model was binary, where "Class = 1" indicated a fraudulent transaction and "Class = 0" indicated a non-fraudulent transaction.

In terms of data pre-processing, the dataset was first checked for any null (missing) or incorrect values. The data was then shuffled and split into training and testing datasets using "StratifiedShuffleSplit" to ensure the ratio of fraud to non-fraud transactions remained consistent across both datasets. Given the highly imbalanced nature of the dataset, techniques such as random under-sampling and Synthetic Minority Over-sampling Technique (SMOTE) over-sampling were used to create a balanced dataset for training. Under-sampling involved down-sampling the majority class (non-fraudulent) to match the number of fraudulent transactions, thus helping the model learn equally from both classes without bias.

The main idea behind the SMOTE algorithm is to generate synthetic data points of the minority class by interpolating between the minority class instances. In other words, SMOTE creates new data artificially. To achieve this, SMOTE randomly selects a minority class instance and then finds its k nearest minority class neighbors. It then generates new synthetic instances by interpolating between the original minority instance and its k-nearest neighbors.

9. Loss Function

The loss function implemented in this project is the focal loss, which is particularly tailored to address classification problems with imbalanced datasets. Focal Loss (FL) is an upgraded form of Cross-Entropy Loss (CE) that tries to handle the class imbalance problem by assigning more weights to easily misclassified examples. Traditional loss functions like cross-entropy may not perform well because they treat all mis-classifications equally, regardless of the rarity of the class. Focal Loss addresses this by down-weighting the loss applied to well-classified cases and emphasizing hard, misclassified examples. It introduces a modulating element that minimizes the loss contribution from simple samples that are already accurately identified with high confidence. This allows it to focus more on challenging examples that are misclassified or are on the decision border.

In randomly under-sampled dataset, instances from the majority class are randomly removed to balance the class distribution. Focal Loss can help in addressing the information loss caused by under-sampling. By focusing on hard-to-classify examples, it ensures that the model pays more attention to the remaining instances of the minority class. This can lead to better performance compared to traditional loss functions, which might not effectively handle the class imbalance.

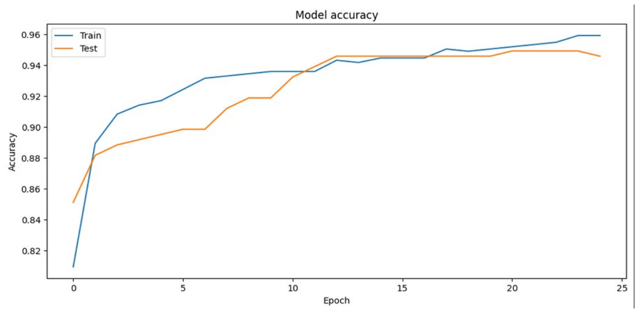

Figure 4: model accuracy (train/test) on under-sampled dataset.

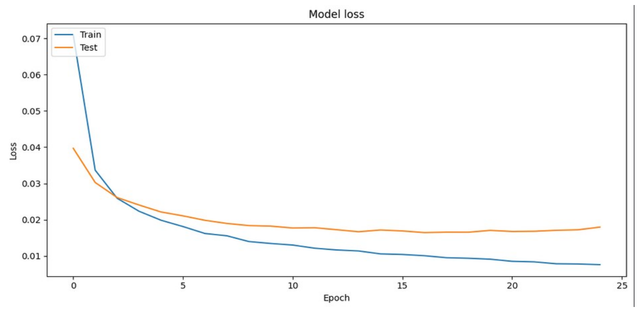

Figure 5: model loss (train/test) on the under-sampled dataset.

The accuracy and loss graphs show the model's performance over 25 epochs of training. From the accuracy graph, we can observe that the model's accuracy on the training set starts higher than 75% and quickly improves, reaching above 95% by the end of the training. The test accuracy, which is the model's performance on unseen data, also shows significant improvement and closely follows the training accuracy. This close correspondence suggests that the model generalizes well and is not overfitting to the training data. The final validation accuracy of about 94.59% indicates a robust model that is able to recognize patterns. The loss graph presents a similarly positive picture. The model starts with a higher loss on the training data, which sharply decreases and then plateaus as the epochs progress, indicating that the model is learning and minimizing the focal loss effectively. The validation loss decreases alongside the training loss and stays consistently close throughout the training process.

Focal Loss can be beneficial in SMOTE over-sampled datasets by focusing on mis-classifications and difficult examples, even among the synthetic instances generated by SMOTE. This ensures that the model learns to discriminate effectively between the classes, including both original and synthetic instances. SMOTE can potentially introduce noise or outliers in the synthetic samples, which might impact the model's performance. Focal Loss can help in mitigating the influence of these noisy samples by down-weighting their contribution to the overall loss, thus improving model robustness.

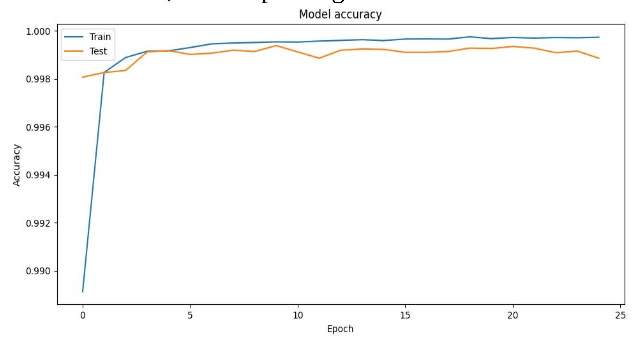

Figure 6: model accuracy (train/test) on over-sampled dataset.

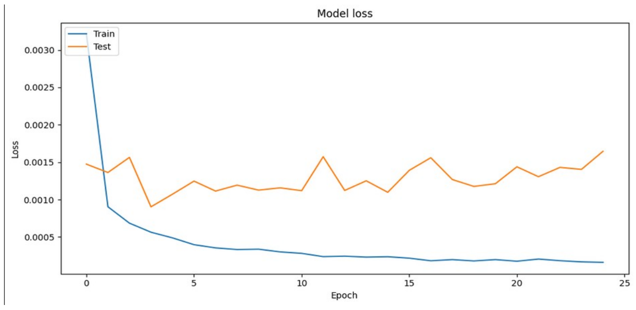

Figure 7: model loss (train/test) on the over-sampled dataset.

In the provided accuracy graph, we observe that the model achieves an extremely high training accuracy, starting higher than 90% and reaching almost 100% accuracy as training progresses. Such high figures might initially seem impressive. However, they could also indicate that the model is learning to fit not only the genuine patterns but also the noise inherent in the synthetically over-sampled data. This is a known risk when using techniques like SMOTE for oversampling. The test accuracy, representing the model's performance on unseen data, remains remarkably stable around 99%, which is encouraging as it suggests the model retains a high degree of generalization when presented with new data. The loss graph complements this view, showing a training loss that decreases sharply in the initial epochs and then levels off, maintaining very low values throughout the remainder of the training process.

10. Model Over-fit Issues and Generalization

The concern of model over-fitting and generalization were addressed through various methodologies. There were several indications of model over-fitting such as high performance on the training data. The models demonstrated high accuracy and other metric scores on the training data. For example, the Logistic Regression model achieved high scores across multiple metrics during the training phase followed by other models such as KNN and SVM. High accuracy, precision, recall, and F1 scores were observed when models were evaluated with the training data, which suggests the models may have been closely fitted to the training dataset characteristics, including any noise present. However, the models were also tested using cross-validation methods and on separate test datasets (both under-sampled and using SMOTE), which showed improved performance.

The project employed cross-validation techniques, such as Stratified Shuffle Split and RandomizedSearchCV/GridSearchCV, which help in assessing how the model would perform on unseen data. Cross-validation is a robust method for estimating the generalization error. The project didn’t rely solely on accuracy due to the imbalanced nature of the dataset. Instead, it also used precision, recall, F1-score, and AUC-ROC to provide a holistic view of model performance. These metrics are crucial in scenarios where one class (fraud) is much rarer compared to the other, as they help in understanding the trade-offs between catching as many frauds as possible (high recall) and not mislabeling non-fraudulent transactions as fraudulent (high precision).

11. Model Hyper-parameters

The hyperparameters for each model were defined to optimize their performance by adjusting their behavior to better fit the training data and improve prediction accuracy on unseen data. These hyperparameters vary by model type and their selection is crucial for the model's ability to effectively learn and make predictions.

A. Logistic Regression: The hyperparameters included “penalty” (type of regularization) and

"C" (inverse of regularization strength). Penalty options included 'l1' (Lasso) and 'l2' (Ridge). Lower 'C' values specify stronger regularization. Regularization helps prevent the model from overfitting by penalizing overly complex models. The "max_iter" parameter was set to 1000 to allow the algorithm more iterations to converge on the optimal solution.

B. K-Nearest Neighbors: The main hyperparameter was "n_neighbors", which determines the number of neighbors to consider for grouping.

C. Support Vector Machines: Hyper-parameters were "C" (regularization parameter) and "kernel" type, which affects the shape of the decision boundary. Different kernels can model non-linear decision boundaries.

D. Decision Trees: Hyperparameters included "criterion" (measure for splitting), "max_depth" (maximum depth of the tree), and "min_samples_lead" (minimum samples a leaf node must have). These parameters help control the complexity of the decision tree, impacting both performance and the risk of overfitting.

Hyper-parameters were primarily chosen using 2 optimization strategies – RandomizedSearchCV and GridSearchCV. GridSearchCV performs an exhaustive search over specified parameter values for an estimator, optimizing for the best cross-validated performance on the training set. RandomizedSearchCV is a faster approach that samples a given number of candidates from a parameter space with a specified distribution. This method is typically less computationally expensive than GridSearchCV and often yields sufficiently good results, especially when the hyper-parameter space is large.

These hyper-parameters had a tremendous impact on performance. For example, regularization in logistic regression helps prevent the model from overfitting by penalizing overly complex models, thereby improving the model's generalization capabilities. In KNN, "n_neighbors" can affect the bias-variance tradeoff. Too few neighbors can lead to a high-variance model (overfitting), whereas too many can lead to high bias (underfitting). Different SVM kernels can also drastically affect the model's ability to capture complex patterns in the data, while deeper trees can model more complex relationships but are prone to overfitting.

12. Machine Learning Frameworks

This project utilized several popular Python libraries and frameworks to implement and manage the machine learning models for fraud detection. These include Scikit-Learn for data splitting, model training, hyper-parameter tuning, and validation, Pandas for data manipulation & analysis, NumPy for handling arrays & matrices, Matplotlib and Seaborn for data visualization, and Imbalanced-learn for handling imbalanced datasets.

The following existing code models served as starting points and reference materials as I started developing my own models for this project – "In-depth skewed data classification" and "Fraud Detection: SMOTE". From these kernels, key insights were gleaned on the application of SMOTE for balancing datasets and the use of various machine learning models. They also highlighted the importance of using precision, recall, and F1-scores as metrics for evaluating model performance due to the skewed nature of fraud detection datasets. These starting points were instrumental in shaping the initial strategy for model selection, data pre-processing, and evaluation techniques in the project.

References

[1] Ali, Abdulalem, et al. "Financial Fraud Detection Based on Machine Learning: A Systematic Literature Review." *MDPI*, Multidisciplinary Digital Publishing Institute, 26 Sept. 2022, www.mdpi.com/2076-3417/12/19/9637.

[2] Hilal, Waleed, et al. "Financial Fraud: A Review of Anomaly Detection Techniques and Recent Advances." *Expert Systems with Applications*, ScienceDirect, 31 Dec. 2021, www.sciencedirect.com/science/article/pii/S0957417421017164.

[3] Sulaiman, Rejwan Bin, et al. *Review of Machine Learning Approach on Credit Card Fraud Detection*, Springer, 5 May 2022, link.springer.com/content/pdf/10.1007/s44230-022-00004-0.pdf.

[4] Galli, S. (2023, March 20). *Exploring oversampling techniques for imbalanced datasets*. Train in Data's Blog. https://www.blog.trainindata.com/oversampling-techniques-for-imbalanced-data/