Final Report:

AI-Based Fraud Detection in Financial Transactions

Literature Reviews and Data Collection

From Python to Julia: Feature Engineering and ML

By Wang Shenghao

Introduction

Shenghao authors an article to demonstrate model building in Julia, which is a high-level language like Python, but with the fast performance of a low-level language, like C. He builds a financial fraud detection model, and goes through the feature engineering, training, and testing of the model in both Julia and Python.

Summary

Shenghao uses a supervised credit card transaction dataset from Kaggle with 30 columns, 28 of which are obtained by PCA. He begins by feature engineering, starting with splitting his data, and making sure that it is stratified. Next, he performs a standard scaling of the data. According to him, scaling helps improve network convergence and prevents an individual feature dominating during training. Then he uses a technique called SMOTE to oversample the minority fraud class. The data is heavily imbalanced in favour of the negative class, and SMOTE needs to be used to synthetically create data for the positive class. Finally, he trains an XGBoost model and tests its precision and recall. He finds that the Julia implementation takes longer to train than the Python one, but it displayed slightly better metrics.

Analysis and Takeaways

A major challenge in financial fraud detection is class imbalance. In these datasets, there are huge imbalances, on the scale of 500:1 in favour of the negative class. It is imperative that counter strategy is employed during the data processing step. We intend to use SMOTE, just like the author, to generate synthetic samples of our minority class. Both Shenghao's and our datasets are similar, so we think it will be a necessary step for our project. We will also employ his strategy of scaling the data and stratifying it during splitting. In conclusion, this article gave our team a great foundation for data preprocessing: namely to stratify split, scale, and resample using SMOTE.

Credit Card Fraud Detection in Python

By Usevalad Ulyanovich

Introduction

Credit card fraud detection has become a critical application of machine learning, addressing the substantial economic losses businesses face due to fraudulent transactions. The guide by Fively explores how Python's rich ecosystem of libraries and tools supports the development of effective fraud detection models, specifically for financial, healthcare, and e-commerce

Summary

The guide outlines a structured approach to building fraud detection models using Python, starting with data preparation and analysis. Techniques like Exploratory Data Analysis (EDA) and dataset partitioning (train-test split) are utilized to identify patterns that differentiate fraudulent transactions from legitimate ones.

Model selection involves using six classification algorithms—K-Nearest Neighbors (KNN), Logistic Regression, Support Vector Machine (SVM), Random Forests, XGBoost, and Decision Tree—to predict fraudulent activity. These models are then evaluated using metrics such as Accuracy, F1 Score, and Confusion Matrix to ensure robustness and precision. Python libraries like Scikit-learn and XGBoost play a crucial role in efficient model development and testing, while additional tools enhance visualization and data manipulation. Furthermore, the blog highlights Python's versatility in automation, web development, and data analysis, showcasing its ability to create scalable and adaptable solutions for fraud detection.

Analysis and Takeaways

The methodology presented highlights Python's versatility and underscores the significance of a data-driven approach to fraud detection. By incorporating various classification models, the approach allows for performance comparison, while the use of evaluation metrics ensures actionable insights. The methodology's strengths include comprehensive coverage of fraud detection processes, a clear explanation of Python's role in simplifying complex tasks, and practical code examples for implementation. However, there are notable limitations, such as a reliance on high-quality data, which may not be accessible to all organizations, and the need for advanced technical expertise to build and maintain the models.

Dataset Preparation

In the data collection phase of our financial fraud detection project, we encountered a rich dataset that provided valuable insights into transaction patterns. However, as with any real-world data, several challenges emerged during the data handling process. This section outlines the steps and decisions we took to properly prepare the data for analysis and use in model training/prediction.

Data Characteristics and Challenges

Class Imbalance

One of the most significant challenges we faced was the severe class imbalance in the fraud labels. The 'isFraud' column in our dataset revealed a disproportionate distribution between fraudulent and non-fraudulent transactions[1]. This imbalance is typical in fraud detection scenarios, where legitimate transactions far outnumber fraudulent ones. Such skewness can lead to biased models that perform poorly in detecting actual fraud cases.

Feature Engineering Opportunities

Our data preprocessing revealed opportunities for feature engineering. We created new features such as 'senderAccChangeRate' and 'recipientAccChangeRate' to capture the rate of change in account balances[1].

Addressing the Challenges

To tackle these challenges, we implemented a multi-faceted approach:

1. For class imbalance, we explored techniques such as upsampling our split data by using SMOTE.
2. To address scalability, we picked RobustScaler due to it being more appropriate when scaling datasets that are dependent on outliers. This decision was a recommendation gained from our Literature Review.
3. Our feature engineering process involved creating new, informative features and normalizing existing ones to capture subtle patterns in transaction behavior.
4. For categorical data, we employed one-hot encoding, transforming the 'paymentType' into binary features for each category.
5. We also label-encoded the sender and recipient account columns. It wouldn’t be reasonable or efficient to one-hot encode them due to the shear size of these categorical columns. Then, we dropped the original column to save some memory.

Model Training

The models we have employed are: Logistic Regression, K-Nearest Neighbors (KNN), Decision Trees, Random Forest, and Deep Neural Network (DNN). Although mentioned in the literature review, the support Vector Classifier (SVC) will not be discussed in this report as it took far more resources (time, CPU cores, RAM, …) to train.

Static Parameters

The common static parameter across all traditional models was class_weights. Due to the imbalanced nature of the problem, we gave classes weightings proportional to their frequencies in the test set. Class 1 had a weighting of 0.998 and class 0 had a weight of 0.002. These weights were derived based on class frequency and then normalized. The reason we used the test set frequencies was because the training set was up sampled and had lost the imbalanced nature.

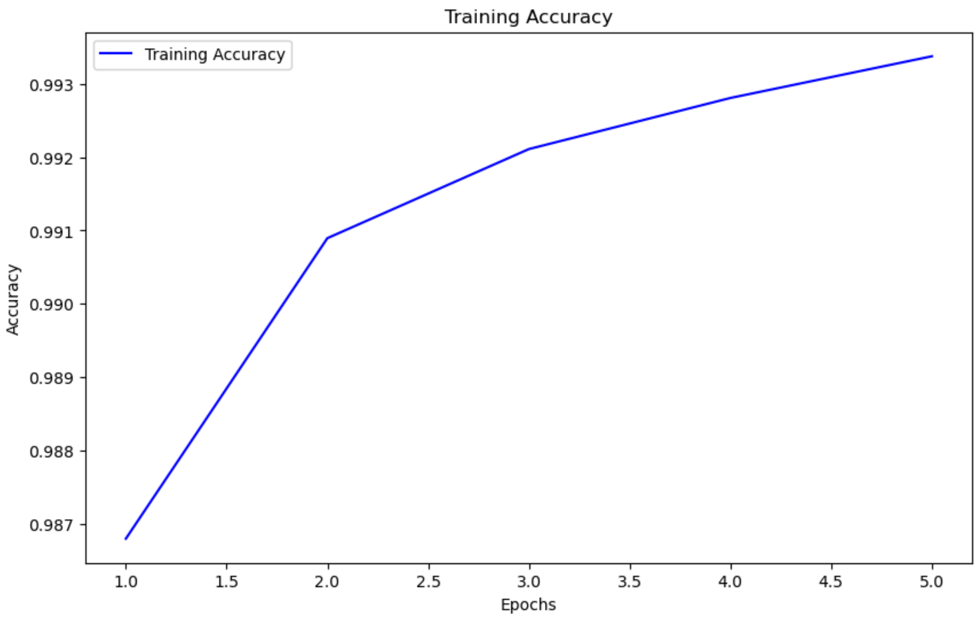
Hyperparameters Grid Searched

Algorithm	**Hyperparameters Searched**	**Values**	**Optimal**
Logistic Regression	C	0.001, 0.01, 0.1, 1, 10	1
	max_iter	100, 200, 300	300
Random Forest	n_estimators	50, 175, 300	300
	max_depth	None, 10, 20, 30	None
Decision Tree	max_depth	10, 20, 30, 40	0.1
	min_samples_spit	2, 5, 10	2
KNN	n_neighbors	5, 7, 9, ..., 48, 50	40
	weights	uniform, distance	distance

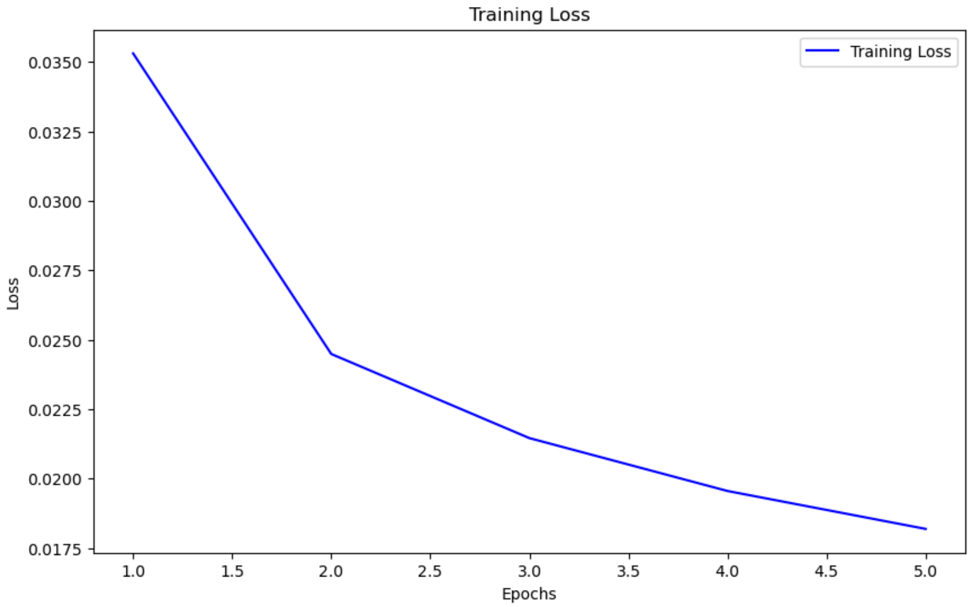
DNN Model

Model Architecture	1. Input Layer: 64 neurons, ReLU activation, input dimension matching the scaled dataset features. 2. Hidden Layer: Single hidden layer with 32 neurons and ReLU activation.

	3. Output Layer: Single neuron with sigmoid activation, suitable for binary classification.
Model Compilation	Optimizer: Adam optimizer with a learning rate of 0.001 Loss Function: Binary cross-entropy Metrics: Accuracy
Training Process	- Epochs: 5 - Batch Size: 64 - Validation Data: test validation during training Training Results: - Epoch 1: - Training accuracy: 0.9802, Validation accuracy: 0.9833 - Epoch 5: - Training accuracy: 0.9935, Validation accuracy: 0.9887
Model Evaluation	Test Accuracy: 99.90% - Majority Class (non-fraudulent transactions): - Precision: 1.00, Recall: 1.00, F1-score: 1.00 - Minority Class (fraudulent transactions): - Precision: 0.59, Recall: 0.86, F1-score: 0.70



Training Accuracy
Training Accuracy
Accuracy
0.993
0.992
0.991
0.990
0.989
0.988
0.987
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Epochs



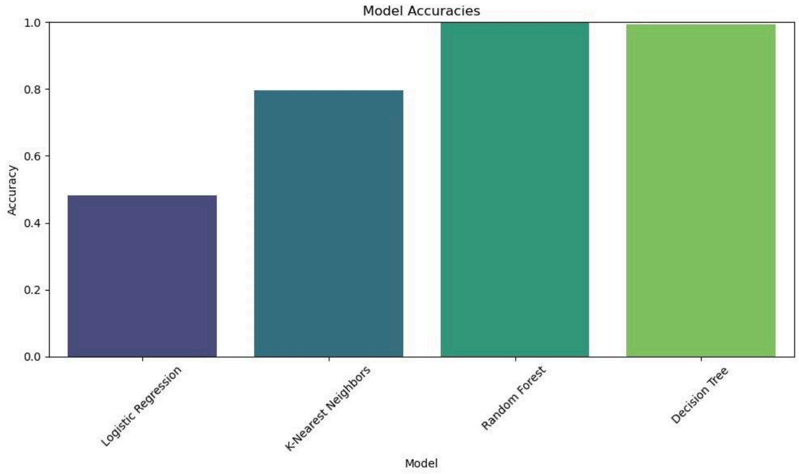
Training Loss
Training Loss
Loss
0.0350
0.0325
0.0300
0.0275
0.0250
0.0225
0.0200
0.0175
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Epochs

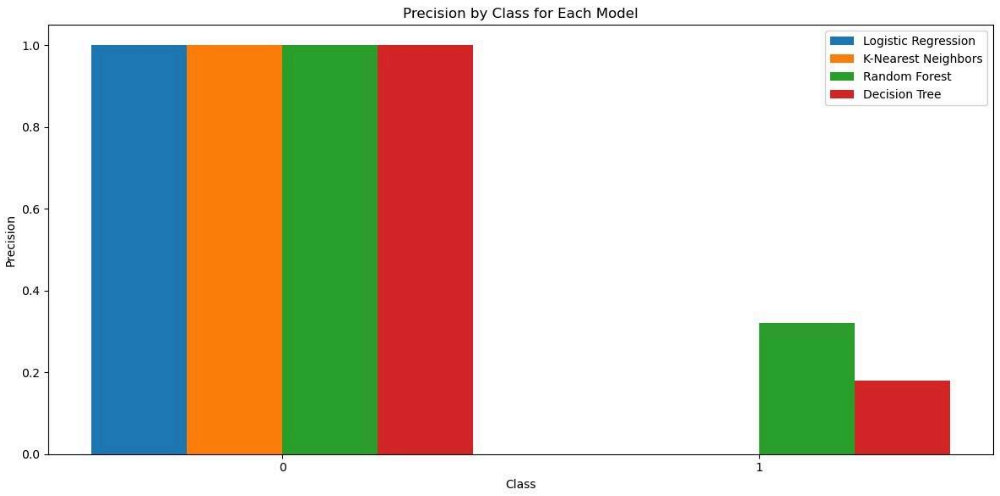
Phase 4: Testing

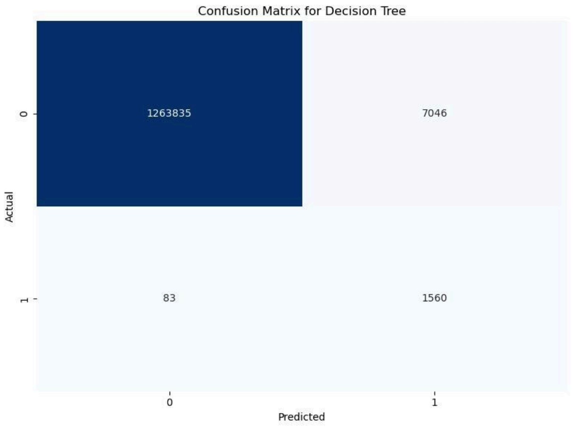
The metrics we used were Accuracy, Recall, Precision and F1 Score. We chose these metrics because our problem was classification. To evaluate these metrics, we ran predictions on the test set, then computed metrics based on the built-in function classification_report.

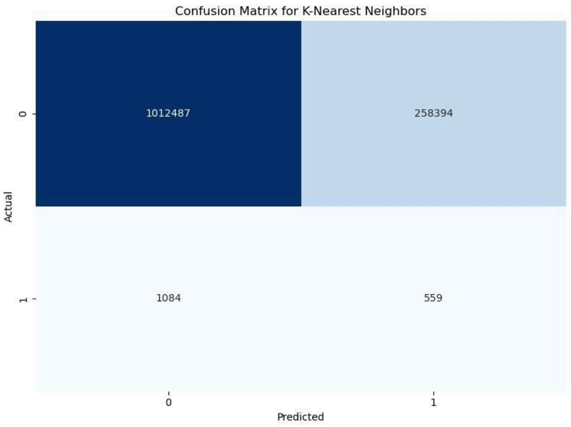
Evaluation Results

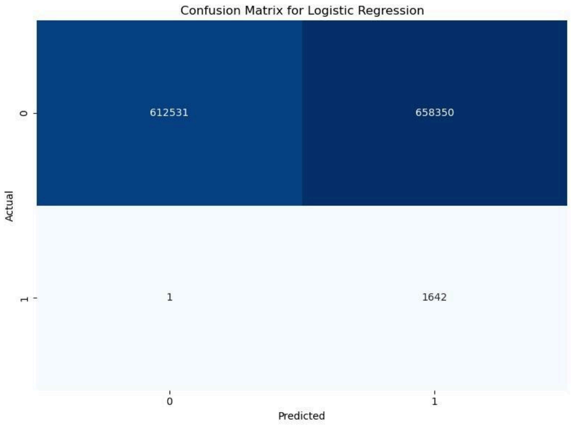
Class	Metrics	Log. Reg.	KNN	DT	RF	DNN
0	Precision	1	1	1	1	1
	Recall	0.48	0.8	0.99	1	1
	F1 Score	0.65	0.89	1	1	1
1	Precision	0	0	0.18	0.32	0.62
	Recall	1	0.34	0.95	0.93	0.87
	F1 Score	0	0	0.3	0.48	0.73
Accuracy		0.48	0.8	0.99	1	1

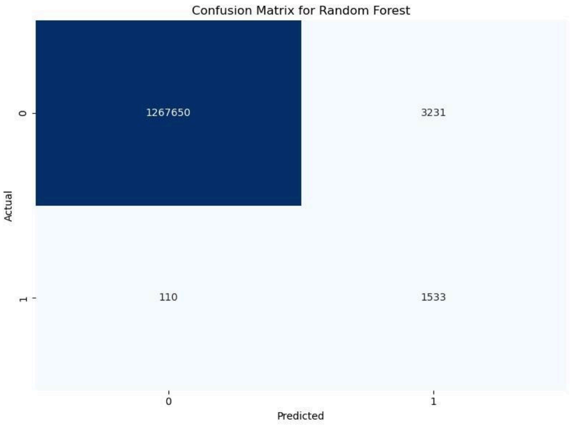



Precision by Class for Each Model
Logistic Regression
K-Nearest Neighbors
Random Forest
Decision Tree
Precision
1.0
0.8
0.6
0.4
0.2
0.0
0
1
Class


Confusion Matrix for Decision Tree
Actual
0
1
1263835
7046
83
1560
0
1
Predicted


Confusion Matrix for K-Nearest Neighbors
Actual
0
1
1012487
258394
1084
559
0
1
Predicted


Confusion Matrix for Logistic Regression
Actual
0
1
612531
658350
1
1642
0
1
Predicted


Confusion Matrix for Random Forest
Actual
0
1
1267650
3231
110
1533
0
1
Predicted

Interpretation

All our models had high metrics for class 0, which is legitimate transactions. This is because the dataset was so imbalanced, that the model learned it was more cost-effective to classify transactions as legitimate than to guess a transaction was fraudulent. When it came to class 1, most traditional models performed abysmally. The models have high recall but low precision, meaning that they would just classify everything as 0. The models were good at predicting the 0 class but struggled to predict the 1 class well.

The only exception to this was the DNN model, which performed excellently on class 0, and decently on class 1. Looking at the class 1 stats for the DNN, we can see that it far outperformed its traditional competitors.

Discussion

Interpretation of Results

The project demonstrated that handling class imbalance is critical in financial fraud detection. Traditional models such as Logistic Regression, KNN, and Decision Trees performed well on the majority class, accurately identifying legitimate transactions. However, these models struggled to generalize to the minority class, resulting in poor precision and recall for fraudulent transactions. This limitation highlights the inherent challenges of imbalanced datasets, where the majority class can dominate the learning process. The Deep Neural Network (DNN), on the other hand, outperformed traditional models by effectively leveraging advanced techniques such as threshold optimization, synthetic data augmentation via SMOTE, and a well-designed architecture tailored to the dataset. These strategies allowed the DNN to improve detection rates for fraudulent transactions without compromising performance on legitimate transactions, thereby striking a better balance between precision and recall.

Comparison with Literature

Our results deviated from our literature. In the literature, Wengshao was able to get much higher testing scores using traditional models. We believe the reason for this may lie in the extra data processing steps he took. He added additional features like transaction counts and step. We believe these features may have played a role in contributing to his improved model performance.

Lessons Learned

Several key lessons emerged from this project. First, addressing class imbalance early in the preprocessing phase is essential for developing robust fraud detection models. Techniques like SMOTE not only balanced the dataset but also improved the ability of models to detect minority class instances. During the project, we gained an important insight into handling class imbalance in fraud detection. Initially, during the development of the DNN model, we achieved 99% accuracy, with strong metrics for class 0 (non-fraudulent), but very low precision and F1-score for class 1 (fraudulent). Although the model's overall accuracy was 99%, this highlighted the impact of imbalanced data on performance. To address this, we adjusted the model's sigmoid activation threshold to 0.99, classifying probabilities above 0.99 as fraudulent. This fine-tuning significantly improved the model's precision and F1-score for class 1 while keeping accuracy, demonstrating the critical role of threshold optimization in tackling imbalanced classification problems.

Second, there is a trade-off between the simplicity of traditional models and the superior performance of more complex models like DNNs. While traditional models are easier to interpret and require fewer computational resources, they often fall short in capturing nuanced patterns in imbalanced datasets. Finally, fine-tuning thresholds proved to be a critical step in optimizing performance for the minority class. Adjusting the DNN's sigmoid activation threshold to 0.99 significantly improved its precision and F1-score for fraudulent transactions, demonstrating the importance of this step in real-world applications.

Conclusion and Future Directions

Achievements

This project successfully demonstrated the application of machine learning techniques to tackle financial fraud detection challenges. By implementing robust preprocessing methods such as SMOTE and feature scaling, we addressed the issue of class imbalance and improved model performance. Traditional models like Random Forest and Logistic Regression showcased strong results for legitimate transactions, but the DNN excelled in detecting fraudulent transactions, achieving high precision and recall for the minority class. This underscores the potential of deep learning in handling complex datasets and imbalanced classification problems.

The success of the project also highlighted the effectiveness of tailored model architecture and parameter optimization. Fine-tuning the DNN, particularly through threshold adjustment, allowed us to better balance false positives and false negatives, demonstrating the value of thoughtful calibration in classification tasks. Overall, the project provided a clear framework for addressing the unique challenges of fraud detection while leveraging state-of-the-art machine learning techniques.

Recommendations

To further enhance the scope and performance of fraud detection systems, several directions for future research are recommended. First, ensemble methods that combine the strengths of multiple models could be explored to improve overall robustness and predictive accuracy. For example, blending traditional models with deep learning approaches could yield complementary advantages and mitigate individual model limitations.

Second, the development of real-time fraud detection pipelines is essential for practical applications. These pipelines should integrate scalable infrastructure to handle large volumes of data efficiently and provide instantaneous predictions in production environments. Incorporating distributed computing techniques and cloud-based solutions could ensure that the system remains responsive under high workloads.

Lastly, incorporating additional behavioral features into the dataset could significantly enhance predictive power. Features such as transaction frequency, geographic patterns, or temporal trends may provide deeper insights into fraudulent behaviors. Advanced feature engineering and domain-specific data collection would further refine the models and improve their ability to detect subtle anomalies indicative of fraud. By pursuing these avenues, future efforts can build on the foundational achievements of this project and create more robust, adaptable fraud detection systems.

References

Shenghao, W. (2023, June 27). *From python to julia: Feature engineering and ML*. Medium.https://towardsdatascience.com/from-python-to-julia-feature-engineering-and-ml-d55e8321f888

Ulyanovich, U. (2022, August 19). 🔐 A Guide to Credit Card Fraud Detection in Python. https://5ly.co/blog/fraud-detection-in-python/