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

lgorithm	yperparameters Searched	alues	ptimal
ogistic Regression	C	.001, 0.01, 0.1, 1, 10	I
	nax_iter	100, 200, 300	00
andom Forest	_estimators	0, 175, 300	00
	nax_depth	one, 10, 20, 30	one
ecision Tree	max_depth	10, 20, 30, 40	.1
	nin_samples_spit	, 5, 10	
NN	_neighbors	, 7, 9, ..., 48, 50	0
	veights	niform, distance	istance

DNN Model

odel Architecture	. Input Layer: 64 neurons, ReLU activation, input dimension matching he scaled dataset features. . Hidden Layer: Single hidden layer with 32 neurons and ReLU ctivation.

	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 Evalutation	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