# Literature Review and Data Collection

## Literature Review 1

**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.

## Literature Review 2

**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.

## 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/>

## 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.

#### Missing Values and Data Quality

While our initial examination didn't reveal explicit missing values, we observed instances where transaction amounts or account balances were zero[1]. These zero values, particularly in the 'oldBalanceRecipient' and 'newBalanceRecipient' columns, could represent either genuine zero balances or potential data quality issues. Distinguishing between these scenarios became crucial for maintaining the integrity of our analysis.

#### Scalability Concerns

The dataset's size and complexity presented scalability challenges. With multiple features for each transaction, including step, amount, and various account details, processing and analyzing this volume of data required careful consideration of computational resources[1]. This was particularly important as we aimed to develop a model capable of real-time fraud detection in a production environment, not limiting it to our local testing and analysising environment.

#### 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]. These derived features keep track of the magnitude of the transaction relative to the users’ accounts, with the aim to enhance the model's ability to detect unusual transaction patterns indicative of fraud.

#### Categorical Data Handling

The 'paymentType' column presented as a categorical variable with multiple categories (PAYMENT, TRANSFER, CASH\_OUT, DEBIT)[1]. Effectively encoding this information without introducing bias or losing important distinctions between categories became a key consideration in our preprocessing pipeline. Subsequently, we dropped the original column in order to reduce the amount of memory being used.

#### 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. We conducted thorough data quality checks, investigating zero values and potential outliers to ensure data integrity.
3. 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.
4. Our feature engineering process involved creating new, informative features and normalizing existing ones to capture subtle patterns in transaction behavior.
5. For categorical data, we employed one-hot encoding, transforming the 'paymentType' into binary features for each category.
6. 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.

## References

[1] <https://github.com/dan-p-steven/prepstone-final/blob/main/p3_dnn_template.ipynb>

# Model Design and Prototype Development

Our approach to financial fraud detection involved designing and implementing a multi-model strategy, leveraging six distinct machine learning algorithms. This comprehensive approach allows us to compare and contrast different methodologies, ultimately aiming to achieve the most robust and accurate fraud detection system.

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) would not be discussed in this report as it took far more resources (time, cpu cores, RAM…) to train.

Below, we detail the design and development of each model.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Data Preprocessing** | **Model Architecture** | **Model Compilation** | **Training Process** | **Model Evaluation** |
| Logistic Regression |  |  |  |  |  |
| K-Nearest Neighbors (KNN) |  |  |  |  |  |
| Decision Trees |  |  |  |  |  |
| Random Forest |  |  |  |  |  |
| Deep Neural Network (MLP using TensorFlow and Keras) | 1. Separated features (X) and labels (y) from the dataset. 2. Features scaled using `StandardScaler` to normalize input data, crucial for neural network performance[1]. | 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. | - Optimizer: Adam optimizer with a learning rate of 0.001 - Loss Function: Binary cross-entropy, appropriate for binary classification. - Metrics: Accuracy | - Epochs: 5 - Batch Size: 64 - Validation Data: A separate test set used for 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 | - Test Accuracy: 99.90% Classification Report: - 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 |

Citations:

[1] <https://github.com/dan-p-steven/prepstone-final/blob/main/bestmodel-hb.ipynb>

# Testing and Analysis

Outline the testing procedures and evaluation metrics

Analyze the model's performance and limitations

# Discussion

Interpret the results in the context of financial fraud detection

Compare your findings with existing literature

# Lessons Learned

Reflect on challenges faced and how they were overcome

Discuss insights gained during the project

# Conclusion and Future Directions

Summarize key achievements and contributions

Suggest areas for future research and improvements