Credit Card Fraud Anomaly Detection

Authors: Shveta Sharma, Varshita Kyal & Ujjwal Upadhayay

Abstract

Anomaly detection is a critical task in the financial industry, particularly in detecting fraudulent transactions in credit card data. This report proposes an approach to detect anomalies in credit card data using machine learning techniques. Credit card data is widely used for online and offline transactions, making it susceptible to fraud, and detecting anomalies can help minimize financial losses.

The proposed approach involves several steps. First, the data is preprocessed by removing missing values and outliers, which can lead to biased results. Next, dimensionality reduction techniques, such as Principal Component Analysis (PCA) are used to reduce the number of features in the data. This helps improve the efficiency of the anomaly detection algorithm and reduces the computational time required for training the model.

After preprocessing, supervised learning algorithms such as LSTM, MLP, and ANN are used to identify anomalies in credit card data. The performance of different algorithms is compared, and the best algorithm is selected based on its ability to accurately detect anomalies. The selected algorithm is evaluated using metrics such as precision, recall, and F1-score, and the results demonstrate the effectiveness of the proposed approach in accurately identifying anomalies in credit card data.

Overall, the proposed approach to anomaly detection for credit card data using machine learning techniques is a valuable contribution to the field of financial fraud detection. The report provides a comprehensive analysis of the approach and its effectiveness, making it a valuable resource for researchers and practitioners in the financial industry. The proposed approach can be extended to other types of financial data to improve security and prevent fraud, contributing to a safer and more secure financial system.

Introduction

Anomaly detection is a crucial task in identifying patterns in data that deviate from expected behavior. Such non-conforming patterns are often referred to as anomalies, outliers, discordant observations, exceptions, aberrations, surprises, peculiarities or contaminants, depending on the application domain. Anomaly detection is utilized in various domains, including fraud detection for credit cards, intrusion detection for cybersecurity, fault detection in safety-critical systems, and military surveillance for enemy activities. The importance of anomaly detection is attributed to the fact that anomalies in data can indicate critical information, which requires prompt attention.

Detecting outliers or anomalies in data has been studied in the statistics community since the 19th century. With time, numerous techniques for anomaly detection have been developed by various research communities. Some of these techniques have been specifically designed for particular application domains, while others are more general.

This report aims to provide a structured and comprehensive overview on anomaly detection applicable to fraud detection for credit cards. By doing so, it aims to enhance understanding of the different directions in which research has been conducted on this topic and how techniques developed in one area can be applied to other domains. It intends to provide insights that aid researchers and practitioners in effectively detecting and preventing anomalous activities in their respective domains.

Background

Anomaly detection is one of the most effective approaches to detecting credit card fraud. Anomaly detection involves identifying unusual patterns in data that do not conform to the expected behaviour of the system. Traditional rule-based approaches to fraud detection are often insufficient, as they rely on predefined rules that may not capture all possible fraudulent activities. Machine learning-based approaches, on the other hand, have shown promise in detecting anomalies in financial data, including credit card transactions.

Machine learning-based approaches to anomaly detection in credit card transactions involve preprocessing the data, reducing the number of features, and using various machine learning algorithms to identify anomalies. Some of the commonly used machine learning algorithms for anomaly detection in credit card transactions include clustering algorithms, density-based methods, and neural networks, among others. These algorithms are trained on historical transaction data to detect patterns and identify potential fraudulent activities in real-time transactions.

The dataset(https://www.kaggle.com/mlg-ulb/creditcardfraud) used for analysis contains information about credit card transactions carried out by European cardholders during a period of two days in September 2013. The dataset consists of 284,807 transactions in total, out of which only 492 were fraudulent. The features V1-V28 are principal components obtained with PCA, while 'Time' and 'Amount' are not transformed. 'Time' represents the seconds elapsed between each transaction in the dataset. 'Amount' represents the transaction amount, and can be used for cost-sensitive learning. The response variable is 'Class', with a value of 1 indicating fraud and 0 indicating no fraud.

This dataset is highly imbalanced, with the positive class (frauds) accounting for only 0.172% of all transactions. In other words, the majority of transactions are legitimate, and the number of fraudulent transactions is very small in comparison. This presents a challenge for building a predictive model since the algorithm will be biased towards the majority class, and it may not be able to detect the minority class (frauds) accurately.

Therefore, special care must be taken when analysing this dataset to ensure that the predictive model does not falsely label legitimate transactions as fraudulent. Techniques such as oversampling, under-sampling, and SMOTE can be used to balance the dataset and improve the accuracy of the model.

Methodology:

We will divide the methodology into two distinct sections: the first section will focus on the data pre-processing methodology, while the second section will concentrate on the modeling methodology.

Data Pre-Processing method:

1. Loading and summarizing data:
We loaded and analyzed the details of dataset. This included checking for any missing values and handling them appropriately. The describe () function is then used to get a summary of the data, including statistical measures like mean, standard deviation, and quartiles.

2. Data cleaning:
After describing the data, the data is cleaned by removing any unnecessary columns like Time and converting any categorical variables into numerical variables. The duplicates are also checked for and handled in dataset. Finally, the data is normalized to ensure that each feature is on the same scale using the Standard Scalar from sklearn library.

3. Visualizing the outliers:
Outliers in the data were identified using boxplots of ‘Amount’ variable. Other features are the PCA, hence, depending on the nature of the data and the goals of the analysis, outliers were handled.

4. Replacing columns like Amount with log-scaled amount column:
The original Amount column is replaced with a new column called amount_box that contains the log of the Amount column. The original Amount column was then dropped.

5. Understanding the data imbalance:
The distribution of the target variable is checked to see if the data is balanced or imbalanced. Visualization techniques like histograms were used to understand the distribution of the target variable i.e. 'Class' (0 for no-fraud and 1 for fraudulent transaction.)

6. Treating the imbalance with SMOTE resampling:
If the data is imbalanced, SMOTE (Synthetic Minority Over-sampling Technique) can be used to oversample the minority class. Before resampling, the training set had 368 fraudulent transactions and 216,043 non-fraudulent transactions. After resampling, the training set had an equal number of fraudulent and non-fraudulent transactions, with 216,043 of each. The SMOTE algorithm was used to create synthetic instances of the minority class (fraudulent transactions) to balance the class distribution and prevent the model from being biased towards the majority class (non-fraudulent transactions).

7. Dimensionality reduction using PCA:
Finally, a PCA (Principal Component Analysis) can be used to reduce the number of dimensions in the data. The number of components to keep is determined based on the amount of variance explained. We observed that 79.8% variance is explained by 8 principal components.The reduced dimensions were initially considered to be used for modeling, but we did not get the optimal performance , so we reverted back to resampled data.

Modelling Methodology

After preprocessing the data, the next step is to split the data into training and testing sets. The training set will be used to train the model, while the testing set will be used to evaluate the model's performance on unseen data. The ratio of the training and testing sets with 80% of the data used for training and 20% for testing.
We tried Logistic Regression model, sequential neural network, Multilayer perceptron (MLP), LSTM model.

1. Logistic Regression Model:
Logistic Regression is a popular machine learning model that is used for binary classification problems. It is a linear model that uses a sigmoid function to transform the linear regression output into a probability value between 0 and 1, representing the likelihood of the instance belonging to the positive class. With this model we got high accuracy in predicting the majority class (0), but a low accuracy in predicting the minority class (1). The precision for class 1 is low, indicating that a significant portion of the positive predictions are false positives. However, the recall for class 1 is high, which means that the model can identify a majority of the actual positive cases. The F1-score for class 1 is also low, indicating that the model's performance on class 1 is poor. Overall, the model needs improvement in correctly identifying the minority class.

2. Sequential Neural Network (ANN) (Baseline Model):
Sequential Neural Network is a type of Artificial Neural Network (ANN) where the neurons are organized in a sequence or a series of layers. To apply a neural network (NN) to the preprocessed data, you can use a deep learning library such as TensorFlow or Keras.
This is a simple neural network model with three layers. The first layer (Dense) has 20 units, uses the ReLU activation function, and takes an input of shape (30,) corresponding to the number of features in the input data.
The second layer also uses the ReLU activation function and has 15 units. Finally, the output layer has a single unit and uses the sigmoid activation function, which is common for binary classification problems.
This model is suitable for a binary classification task, where the goal is to predict the probability of an instance belonging to one of two classes.
The model takes in a 30-dimensional feature vector as input and outputs a single probability value indicating the likelihood of the input belonging to the positive class (i.e., class 1). The weights of the model are initialized using the He normal initialization method, which is a commonly used weight initialization technique in deep

learning. It defines a sequential model with three dense layers. The first two layers have a ReLU activation function, and the last layer has a sigmoid activation function. The input shape of the first layer should match the number of features in the preprocessed data. This code compiles the model using the Adam optimizer, binary cross-entropy loss, and accuracy metric.
This code trains the model on the preprocessed training data for 5 epochs, with a batch size of 32. It also uses the preprocessed test data for validation during training.
These classification metrics indicate that the model has achieved perfect accuracy in predicting the majority class (0), but relatively low recall and F1-score for the minority class (1). Specifically, the model correctly classified all instances of the majority class, but only 73 out of 92 instances of the minority class were correctly classified (resulting in a recall of 0.79). The F1-score of the minority class is also relatively low at 0.36, indicating that the model is not performing well in terms of precision and recall for this class. This suggests that the model may be overfitting to the majority class and failing to learn the patterns in the minority class. We would need tuning of hyperparameters to get a better f-1 score from this model.

3. Hyper tuned Parameters sequential neural network model:

The baseline model is tuned for hyper-parameters. The number of units were modified which has three fully connected Dense layers. The first Dense layer has 32 neurons, uses the 'relu' activation function, and applies the He normal weight initialization technique. It also applies L2 regularization with a regularization rate of 0.0001 to prevent overfitting.

The next layer is a Dropout layer, which randomly drops out a fraction (in this case, 0.1) of the previous layer's outputs during training to further prevent overfitting.

The second Dense layer has 16 neurons, uses the 'relu' activation function, and applies the He normal weight initialization technique. It also applies L2 regularization with a regularization rate of 0.0001 to prevent overfitting.

The next layer is another Dropout layer, which randomly drops out a fraction (in this case, 0.1) of the previous layer's outputs during training to further prevent overfitting.

The last Dense layer has a single neuron, uses the sigmoid activation function, and applies the He normal weight initialization technique. It is used for binary classification problems, where the output of the model represents the probability of the input belonging to class 1.

Overall, this model is designed to address overfitting using dropout regularization and L2 regularization. It uses the 'relu' activation function for the hidden layers and the sigmoid activation function for the output layer as our output is binary. This is our best model and we would explain the results further in result section.

4. LSTM (Long Short-Term Memory):

The first line creates an instance of a Sequential model object. This object allows us to add layers to the model one after the other, in a linear stack.

The next four lines add layers to the model. The first layer is an LSTM layer with 70 neurons, using the hyperbolic tangent activation function ('tanh'). The input shape of this layer is set to the shape of the input data X_train_res, with the number of time steps being the length of the second dimension of the input data, and the number of features being 1. The "return_sequences" parameter is set to True, which means that the output of this layer will be fed into the next LSTM layer.

The second layer is also an LSTM layer with 60 neurons, using the 'tanh' activation function. Since the "return_sequences" parameter was set to False (which is the default value), the output of this layer will be a vector rather than a sequence.

The third layer is a fully connected Dense layer with 50 neurons and the 'tanh' activation function. The final layer is also a Dense layer with a single neuron and the sigmoid activation function. Since this is a binary classification problem, the sigmoid function is appropriate as it outputs a value between 0 and 1, representing the probability of the input belonging to class 1. Overall, this model has two LSTM layers with 70 and 60 neurons respectively, followed by two fully connected Dense layers with 50 and 1 neurons, respectively. The model is trained to predict binary class labels and uses the sigmoid activation function in the final output layer. In this case, the f1-score for class 1 is 0.59. This is lower than the f1-score for class 0, which is 1.00. This means that the model is better at predicting class 0 than class 1. The lower f1-score for class 1 is likely due to a lower precision and/or recall for that class.

LSTM doesn't support parallel computing and it takes more than an hour to run a model. It is a dense web model which negates the idea of having a parsimonious model. We tried different tuning parameters to enhance the f1 score but the model kept on getting denser and the computational time also increased exponentially. Hence, this model was dropped from the evaluation of best models.

5. Multilayer Perceptron (MLP):

The MLP model has three hidden layers, with 80, 70, and 60 neurons in each layer, respectively. The activation function used in each neuron is the hyperbolic tangent function ("tanh"). The optimization algorithm used to train the model is the "adam" solver, which is a stochastic gradient-based optimizer. The maximum number of iterations for the solver is set to 500. We tried different combinations of neurons in each layer to get an optimal test accuracy, precision, and f1-score. We could achieve an f1 score of 75% for Frauds Classes 1 but this model is dense with slow training and high computational cost. MLP is even prone to getting stuck in local minima and tends to overfit. This model took longer time to converge, hence, we went with hyper-parameter tuned sequential neural network. For class "1", the precision of the model is 0.71, which means that 71% of the samples predicted as class "1" are true positives. The recall of class "1" is 0.78, which means that the model correctly identified 78% of the true positives in the dataset. The F1-score of class "1" is 0.75, which is the harmonic mean of precision and recall for class "1".

The overall accuracy of the model is 1.00, which means that the model correctly predicted the class labels for all the samples in the evaluation dataset. The macro-averaged F1-score is 0.87, which is the average of the F1-scores of both classes weighted equally.

Tuning Hyperparameters Training:

Learning Rate: The learning rate we started with was 0.01, which was too high, the optimizer may overshoot the optimal weights and never converge to the minimum loss. Therefore, reducing the learning rate can help the optimizer take smaller steps towards the optimal solution and avoid overshooting or getting stuck in a local minimum. So, in iterations, it was reduced to 0.0001 to get optimal performance.

Batch Size: We opted for reducing the batch size can also improve the generalization ability of the model. This is because smaller batch sizes force the model to learn from more diverse examples at each update, which can help prevent overfitting. The batch size in baseline model was 32 that was increased first to 64, then we reduced it back as it provided a optimal performance.

Epochs: The base model has 5 Epochs. Increasing the number of epochs in a neural network can lead to better training of the model, as it allows for more iterations over the training data, potentially improving the accuracy and reducing the loss. Hence, in tuned model, it was moved up to 10.

Unit Size: Comparing our base model of sequential neural network with 20, 15, and 1 unit in its three layers to one with 32, 16, and 1 unit, we can say that the latter has a higher capacity to learn complex patterns in the data due to the larger number of units in the first two layers. However, this higher capacity comes at the cost of potentially overfitting the training data if not properly regularized. Hence, we applied further regularization techniques to cater for it.

No. of layers: Our base model has 3 dense layers; hence, we increased it to 4 dense layers initially in tuned model, but it was causing overfitting, hence, we went back to 3 dense layers but increased the no. of units to enhance the model's ability to read complex patterns.

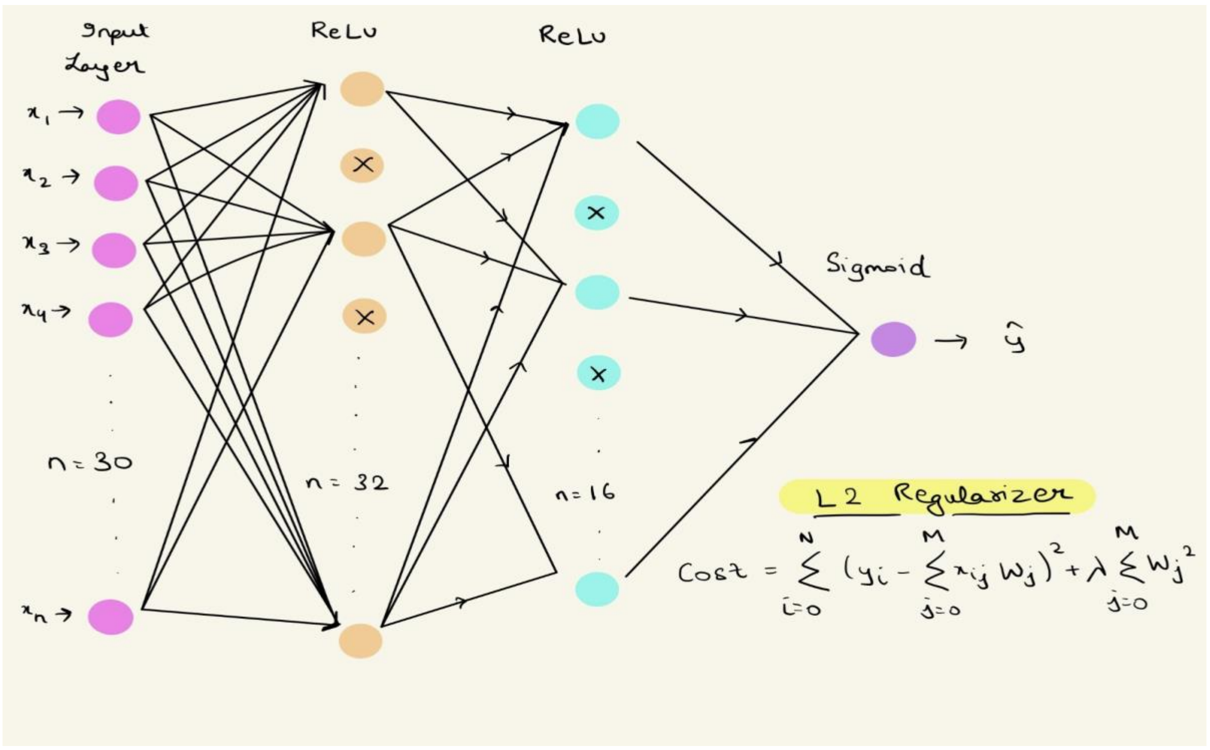
Dropout Layer: Adding a dropout layer with a dropout rate of 0.1 means that during training, 10% of the neurons in the neural network will be randomly dropped out. This can have the effect of making the model more robust and less likely to overfit to the training data.

Regularization: L2 regularization tends to perform better in models where all features are relevant and contribute to the output, while L1 regularization may be more effective in models where there are many irrelevant features that can be pruned.

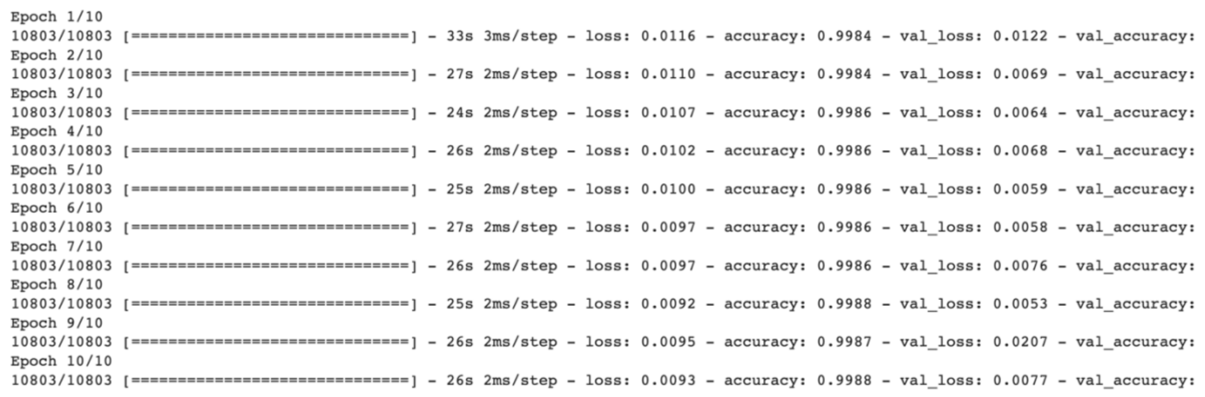
Validation Set: Increasing the size of the validation set from 0.1 to 0.2 means that a larger portion of the data is being used for validation during the training process. So, when we increase the validation set size can lead to better generalization of the model as it is being validated on a larger, more diverse set of data. Hence, we opted for 20% validation set.

Experiments And Results :

Architecture of the Best Model: (Hyper tuned Sequential Neural Network)



Record of Training process of the Best Model:



```
Epoch 1/10
10803/10803 [==============================] - 33s 3ms/step - loss: 0.0116 - accuracy: 0.9984 - val_loss: 0.0122 - val_accuracy:
Epoch 2/10
10803/10803 [==============================] - 27s 2ms/step - loss: 0.0110 - accuracy: 0.9984 - val_loss: 0.0069 - val_accuracy:
Epoch 3/10
10803/10803 [==============================] - 24s 2ms/step - loss: 0.0107 - accuracy: 0.9986 - val_loss: 0.0064 - val_accuracy:
Epoch 4/10
10803/10803 [==============================] - 26s 2ms/step - loss: 0.0102 - accuracy: 0.9986 - val_loss: 0.0068 - val_accuracy:
Epoch 5/10
10803/10803 [==============================] - 25s 2ms/step - loss: 0.0100 - accuracy: 0.9986 - val_loss: 0.0059 - val_accuracy:
Epoch 6/10
10803/10803 [==============================] - 27s 2ms/step - loss: 0.0097 - accuracy: 0.9986 - val_loss: 0.0058 - val_accuracy:
Epoch 7/10
10803/10803 [==============================] - 26s 2ms/step - loss: 0.0097 - accuracy: 0.9986 - val_loss: 0.0076 - val_accuracy:
Epoch 8/10
10803/10803 [==============================] - 25s 2ms/step - loss: 0.0092 - accuracy: 0.9988 - val_loss: 0.0053 - val_accuracy:
Epoch 9/10
10803/10803 [==============================] - 26s 2ms/step - loss: 0.0095 - accuracy: 0.9987 - val_loss: 0.0207 - val_accuracy:
Epoch 10/10
10803/10803 [==============================] - 26s 2ms/step - loss: 0.0093 - accuracy: 0.9988 - val_loss: 0.0077 - val_accuracy:
```

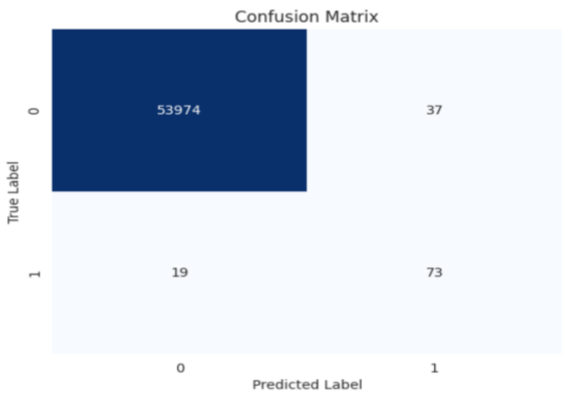
Model Evaluation Metrics:

Table 1.

Model	Precision	Recall	F1-Score	Support
Logistic Regression	0.06	0.85	0.12	92
Sequential Neural Network	0.23	0.79	0.36	92
Hyperparameter tuned ANN	0.66	0.79	0.72	92
LSTM	0.48	0.76	0.59	92
Multilayer Perceptron	0.71	0.78	0.75	92

From Table 1., the performance of different machine learning models can be evaluated based on several metrics such as precision, recall, and F1-score. In the context of a binary classification problem where one class is vastly outnumbered by the other, precision is a crucial metric to consider. It indicates the proportion of true positive predictions over the total positive predictions. On the other hand, the F1-score provides a balance between precision and recall, which measures the proportion of true positive predictions over the total actual positives.
Among the models evaluated, the hyperparameter tuned sequential neural network stands out with the high precision score of 0.66, indicating that it correctly classified a substantial proportion of the positive class. It also achieved the highest F1-score of 0.72, demonstrating an optimal balance between precision and recall. In comparison, the logistic regression model exhibited very low precision and F1-score, while the sequential neural network and LSTM models had better precision but relatively lower F1-scores. The Multilayer Perceptron (MLP) model achieved a precision and F1-score score like the hyperparameter tuned sequential neural network, but given the architecture of MLP, we know the model is complex with dense web of layers, also MLP is prone to get stuck in local minima and tends to overfit. MLP is slow in training and high computational cost. Hence, the model is not parsimonious compared to hyper-tuned sequential neural network. Overall, the hyperparameter tuned sequential neural network proved to be the most effective model in classifying the positive class, outperforming the other models in terms of precision and F1-score.

Confusion Matrix for Best Model:



From the above confusion matrix, we have a misclassification rate of 0.001 which is very low. Hence, our hyper-tuned ANN model is providing an optimal fit for the credit card anomaly detection data. Therefore, we would recommend this parsimonious model due to its less computational cost and highly effective precision and decent F1 score.

References:

1. https://towardsdatascience.com/detect-anomalies-in-telemetry-data-using-principal-component-analysis-98d6dc4bf843
2. https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud
3. https://github.com/atevhs/DATA586_AdvancedML