An Intelligent Approach For Anomaly Detection In Credit Card Data Using Bat Optimization Algorithm

Rossella Carone

Personal code: 10724687

Numerical Analysis for Machine Learning, a.y. 2023-2024

Contents

1 Introduction **2**
- 1.1 Abstract . . . 2
- 1.2 Overview . . . 2
- 1.3 Dataset . . . 2
  - 1.3.1 Exploratory Data Analysis . . . 3

2 Data Preprocessing **5**
- 2.1 Feature Engineering . . . 5
- 2.2 Feature Selection . . . 5
- 2.3 Data Balancing . . . 8
- 2.4 Training, Validation and Test Set . . . 9

3 Model **10**
- 3.1 Architecture . . . 10
- 3.2 Loss Function . . . 11
  - 3.2.1 FCL Function . . . 11
  - 3.2.2 Binary Cross-Entropy Loss Function . . . 12

4 Performance Evaluation **13**
- 4.1 Metrics . . . 13
- 4.2 Model Results . . . 14

5 Conclusions **15**
- 5.1 Weaknesses Of The Paper . . . 15
- 5.2 Final Considerations . . . 15

6 Bibliography **16**

A Complete CNN Model **17**

Introduction

1.1 Abstract

The present work aims to reproduce the results of the paper titled *An intelligent approach for anomaly detection in credit card data using Bat optimization algorithm* [1] and to critically analyze it. The paper addresses the development of a stable credit card fraud detection model that performs feature selection, data balancing and deep representation learning. The review intends to assess the scientific robustness of the paper [1] by identifying its strengths and weaknesses.

1.2 Overview

Credit card fraud detection is a crucial aspect of financial security for both consumers and financial institutions. With the rise of online transactions and digital payment systems, detecting fraudulent activities has become increasingly challenging. One of the biggest challenges in credit card fraud detection is the highly unbalanced nature of the datasets, where the number of legitimate transactions massively outweighs the number of fraudulent transactions. This imbalance between classes can lead to biased models that perform poorly in detecting fraudulent activities.
The proposed approach executes automated fraud detection, uncovering hidden correlations within data. Differently to related works about this topic, this achievement is attained by employing the Bat Optimization Algorithm (BA) to select the relevant features. Additionally, Synthetic Minority Over-sampling Technique (SMOTE) is used to address the imbalance within the dataset. Subsequently, a Artificial Neural Network model is constructed, incorporating a full center loss function to enhance both the performance and stability of fraud detection.
The efficacy of the proposed model is successively tested and demonstrates an accuracy rate of approximately 98%.

1.3 Dataset

The dataset [4] sourced from Kaggle comprises credit card transactions conducted by European cardholders in September 2013. Recorded over a span of two days, it encompasses 492 instances of fraud out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions. It contains 28 numerical input variables resulting from Principal Component Analysis (PCA) and the features *Time*, *Amount* and *Class*. Indeed, the original features and additional background information are unavailable due to confidentiality constraints. *Time* indicates the elapsed seconds between each transaction and the initial transaction in the dataset, whereas *Amount* denotes the transaction amount. The *Class* feature is the response variable, assuming a value of 1 in the event of fraud and 0 otherwise.

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.189115	0.133558	-0.021053	149.62	0.0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	0.125895	-0.008983	0.014724	2.69	0.0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	-0.139097	-0.055353	-0.059752	378.66	0.0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.221929	0.062723	0.061458	123.50	0.0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	0.502292	0.219422	0.215153	69.99	0.0

Figure 1.1: Credit card fraud dataset from Kaggle

1.3.1 Exploratory Data Analysis

This section presents a first exploratory data analysis conducted to gain insights into the dataset's characteristics and identify potential issues that may require further attention.

1. **Data Cleaning:**

   A thorough examination reveals 1081 duplicated records and no missing values. After having removed such repetitions, the dataset consists of 283,726 rows and 31 columns.

2. **Features Distribution:**

   Observing the highly varied distribution in Figure 1.2, the distinct scale exhibited by the variable *Amount* in comparison to the other variables is particularly evident. Hence, data scaling should be performed.

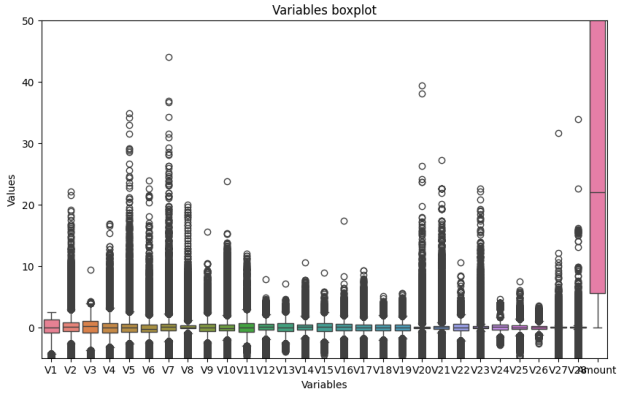

Figure 1.2

3. **Target Distribution**

As shown in Figure 1.3, the presence of only 473 fraudulent credit card transactions out of a total of 283,726 transactions stresses the need for data balancing techniques. Unbalanced datasets, where the occurrence of one class (fraudulent transactions) is significantly lower than the other (non-fraudulent transactions), can lead to biased models that inaccurately prioritize the majority class.

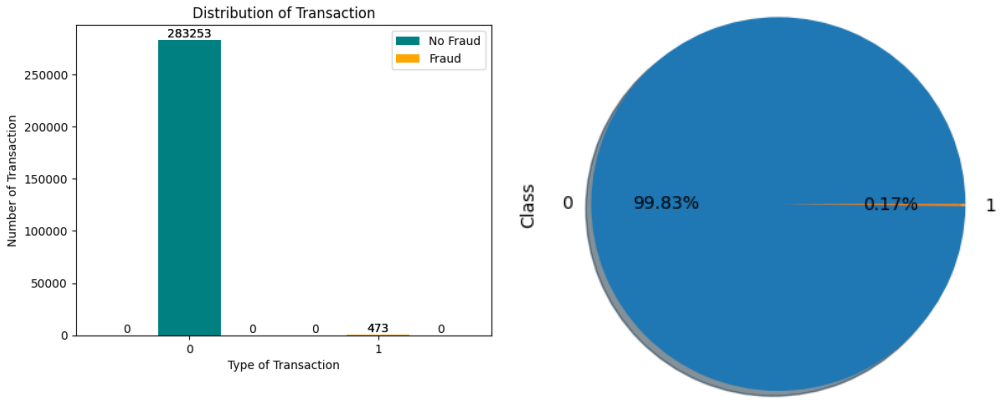

Figure 1.3: Histogram visualization (on the left) and Pie Chart visualization (on the right)

4. **Correlation Analysis**:

The Heatmap in Figure 1.4 is useful to visualize the correlation matrix of variables in the dataset. It provides an intuitive way to identify the direction of associations between different features, which is crucial for feature selection and model building.

- Some variables exhibit negative correlations with the response variable. It means that as these values decrease, the likelihood of a fraudulent transaction increases.
- Other variables instead are positively correlated with the outcome variable. It means that as these values increase, the likelihood of a fraudulent transaction also increases.

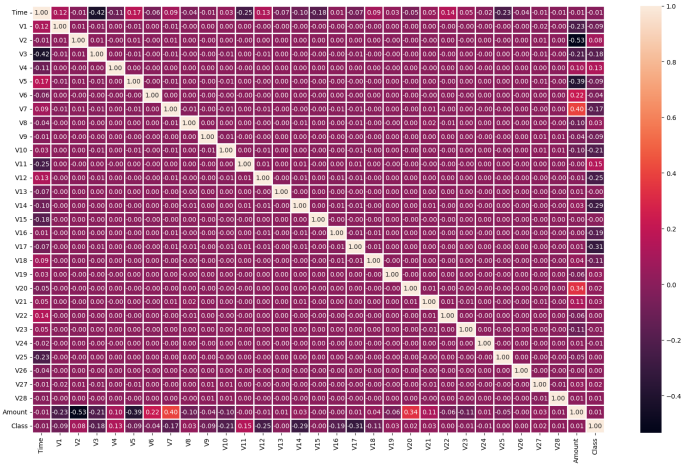

Figure 1.4

Data Preprocessing

Building an effective credit card fraud detection model consists of some essential steps, as discussed in the sections below.

2.1 Feature Engineering

The initial phase typically involves feature engineering, which aims to extract informative attributes from users' transaction behaviors to train predictive models. A common approach involves creating new features through transaction aggregation by grouping them based on factors like time intervals, card numbers, transaction types, and merchant codes. Through this aggregation process, individual transactions are transformed into a feature matrix containing more informative attributes.
Unfortunately, the original features about this dataset are not publicly available for confidentiality issues. Since transactions cannot be associated with the corresponding cardholders' data, it would be impractical to aggregate features for acquiring useful information.

2.2 Feature Selection

Feature selection is a fundamental process in machine learning whose purpose is to identify and choose the most relevant features for modeling while discarding irrelevant or redundant ones.
By doing this, feature selection possibly improves model performance, reduces computational complexity and enhances interpretability. It plays a crucial role when considering the *bias-variance trade-off*.
Feature selection for the present work is fulfilled via **Bat Algorithm** (BA) [2]. This algorithm is designed by idealizing the echolocation characteristics of bats where, for the sake of simplicity, the following assumptions are made:

1. All bats use echolocation to sense distance, and they also know the difference between prey and background barriers;

2. Bats fly randomly with velocity v_i at position x_i with a fixed frequency f_{min},varying wavelength λ and loudness A_0 to search for prey. They can automatically adjust the wavelength (or frequency) of their emitted pulses and adjust the rate of pulse emission p $\in$ [0, 1], depending on the proximity of their target;

3. Although the loudness can vary in many ways, it is assumed that the loudness varies from a large (positive) A_0 to a minimum constant value A_{min}.

Here a step-by-step explanation of the algorithm:

Algorithm: **BA**

Initialize position x_i and velocity v_i of the bat population ($i = 1, ..., N$).
Set the frequency f_i, pulse rate p_i and loudness A_i to their default values.

While (n < Maximum number of iterations)

Generate new solutions by adjusting frequency, and update velocities and locations/solutions.

If ($rand > p_i$)

- Choose a solution from the list of the best solutions
- Generate a local solution around the selected best solution

Create a new solution by flying around at random.

If ($rand < A_i$ & $f(x_i) < f(x^*)$)

- Accept the the new solutions.
- Increase p_i and reduce A_i

end if

Rank the bats and find the current best x^*.

end while

Virtual bats are considered for simulations. In the following, the update rule for frequencies f_i, positions x_i and velocities v_i is presented. The new solutions f_i, x_i^t and v_i^t at time step t are given by:

$$
\begin{aligned}
f_i &= f_{min} + \beta(f_{max} - f_{min}) \\
v_i^t &= v_i^{t-1} + f_i(x_i^t - x^*) \\
x_i^t &= x_i^{t-1} + v_i^t
\end{aligned}
$$

where $\beta \in [0, 1]$ is a random variable drawn from a uniform distribution and x^* is the current global best location identified after comparing all the solutions among all the N bats. As the product $\lambda_i f_i$ is the velocity increment, we can use either f_i (or λ_i) to adjust the velocity change while fixing the other factor, depending on the type of the problem of interest. Initially, each bat is randomly assigned a frequency which is drawn uniformly from $[f_{min}, f_{max}]$.
For the local search part, once a solution is selected among the current best solutions, a new solution for each bat is generated locally using the following random walk:

$$x_{new} = x_{old} + \epsilon A^t$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ (σ is the scaling factor) and A^t is the average loudness of all the N bats at time step t. Furthermore, the loudness A_i and the pulse rate p_i have to be updated accordingly as the iterations proceed. For simplicity, $A_0 = 1$ and $A_{min} = 0$ assuming that $A_{min} = 0$ indicates that a bat has just discovered the

prey and has temporarily stopped generating any sound. Then:

$$A_i^{t+1} = \alpha A_i^t, \qquad p_i^{t+1} = p_i^0(1 - e^{-\gamma t})$$

where α and γ are constants and set equal to 0.9. It can be proved that for any $\alpha \in (0, 1)$ and for any $\gamma > 0$:

$$A_i^t \to 0, \qquad p_i^t \to p_i^0$$

The algorithm is computationally expensive when all observations are considered, therefore only the observations coming from the validation set are considered to perform feature selection. After the application of the Bat Algorithm the following relevant features are identified:

Number of Features: 5

```
Selected Features = ['Time', 'V4', 'V6', V10', 'V17']
```

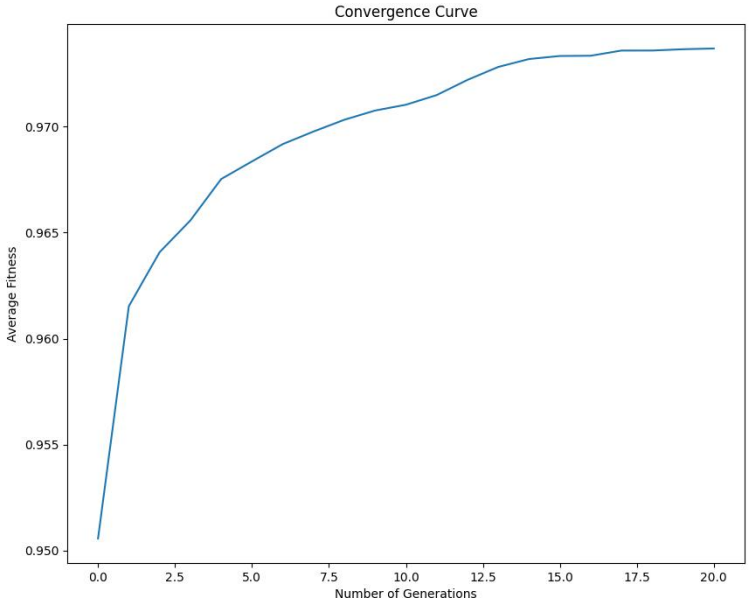

Figure 2.1: Convergence curve of the Bat Algorithm optimization algorithm.

2.3 Data Balancing

The imbalance within the dataset negatively affects the performance of Machine Learning models. Indeed if the class imbalance problem is not considered, the learned classifier will tend to identify most of the fraud transactions as genuine ones. The reason is that almost all classifiers have a default assumption of a balanced data set, and thus, the learned decision boundary tends to bias toward the class with more samples. Hence, dealing with the class imbalance problem has become an indispensable step before training a fraud detection model. To address this issue, **Synthetic Minority Over-sampling Technique** (SMOTE) is employed. SMOTE mitigates the imbalance by generating synthetic samples for the minority class. The process involves selecting a minority class instance at random and finding its k nearest neighbors in the feature space. Synthetic samples are then generated along the line segments connecting these nearest neighbors. As it can be seen in figure below, the class sizes display now a good equilibrium, with 283,253 observations within each class.

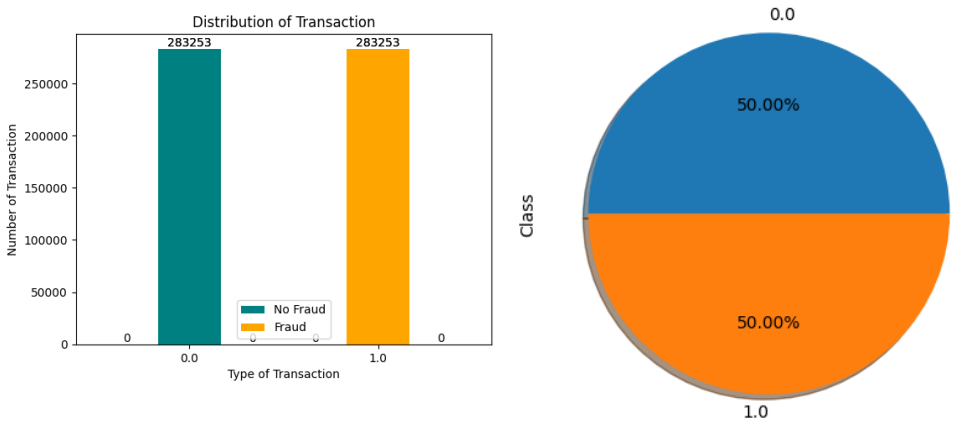

Figure 2.2: Histogram visualization (on the left) and Pie Chart visualization (on the right)

Here's a step-by-step explanation of how SMOTE algorithm works:

Algorithm: **SMOTE algorithm**

Input: Raw training dataset D
Output: Oversampled training davaset D^+

- Calculate the oversampling data size S^+ and initialize D = D^+
- Identification of noise, borderline and edge samples
- Sample importance calculation
  - Calculate borderline sample importance
  - Calculate noisy sample importance
  - Calculate edge sample importance
- Synthetic minority samples generation for j= 1,.., S^+
  - Select one sample from the borderline, edge and noise minority samples an a^+ with respect to their samples importance
  - Select a nearest neighbor of a^+_{knn} with respect to their sample balance

  - Calculate a value with respect to the sample importance of a^+ and a^+_{knn}
  - Generate the synthetic minority samples and add it to D^+;
- Export the oversampled training dataset D^+ to the classification model

---

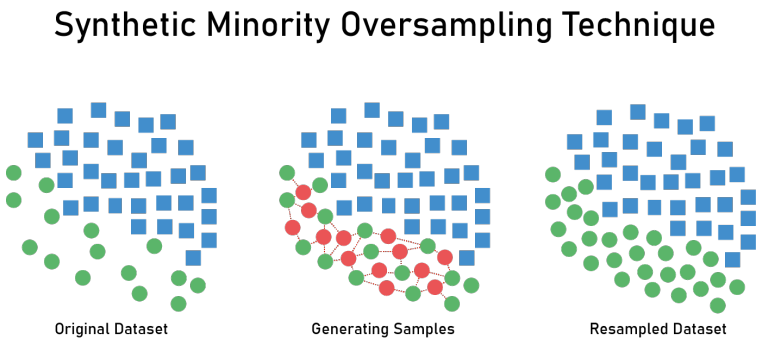

Figure 2.3: Dataset before and after the application of SMOTE algorithm

2.4 Training, Validation and Test Set

A 70:15:15 split is applied to the dataset using the `train_test_split` function. The training set, representing 70% of the total data, is used for model training, while 15% forms the validation set, used for tuning model parameters, and the remaining 15% forms the test set, utilized for evaluating model performance.

Model

3.1 Architecture

The dataset sourced from Kaggle contains no more than 0.5 million transactions with at most 30 available features. Its size, relative to many machine learning datasets, underscores the necessity for careful model selection. Therefore, a fully connected neural network architecture is designed for this case.
It consists of the following layers:

- **Input Layer**: it takes the 5 features derived from the feature selection phase
- **First Hidden Layer**: 32 neurons
- **Second Hidden Layer**: 64 neurons
- **Third Hidden Layer**: 16 neurons
- **Fourth Hidden Layer**: 8 neurons
- **Output Layer**: 1 neuron

Each neuron utilizes the Rectified Linear Unit (ReLU) activation function to introduce non-linearity. The output layer consists of a single neuron with a sigmoid activation function, suitable for binary classification tasks. For training, Stochastic Gradient Descent (SGD) optimizer is employed. SGD is a classical optimization algorithm widely used for training neural networks. It updates the weights of the network by computing the gradient of the loss function with respect to the weights of the network for a small batch of examples.
The architecture depicted in Figure 3.1 is presented here with a reduced number of neurons for better visualization.

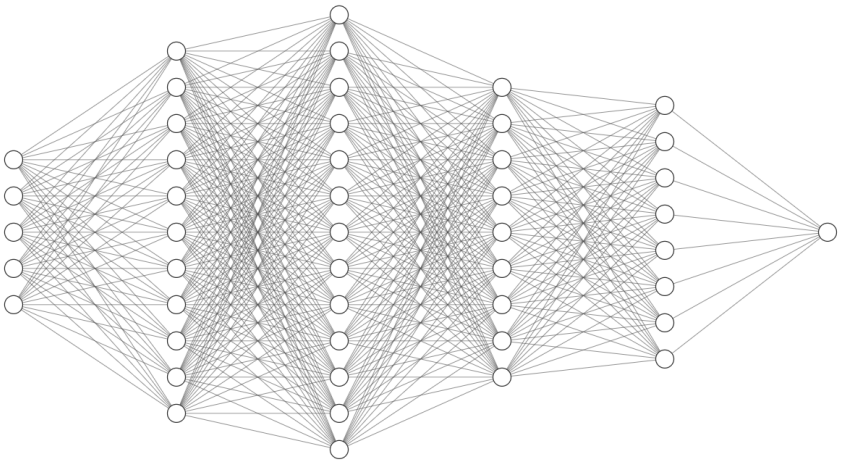

Figure 3.1: Artificial Neural Network

More details about the more complex architecture employed for the other (private) dataset can be found in the appendix A.

3.2 Loss Function

In this model two distinct factors constitute the overall loss function, which is minimized during the training process to improve the model's performance.

1. **Full Center Loss**: it refers to the loss associated with the process of embedding or feature learning and is weighted by a hyperparameter β which controls the contribution of the FCL component to the overall loss function, providing flexibility in optimizing the model's performance.

2. **Binary Cross-Entropy Loss**: it is commonly used in binary classification problems to assess the discrepancy between the predicted probabilities and the actual class labels.

3.2.1 FCL Function

The Full Center Loss Function is introduced to supervise the training of the neural network layers that project the original feature space of transactions into a deep feature space.
The objective is to maximize the grouping of transactions belonging to the same class while simultaneously maximizing the distinction between transactions from different classes.
It combines two types of losses: ACL for dealing with the separability of transactions from different classes and DCL for dealing with the compactness of transactions from the same class.

FCL can be formulated as follows:

$$L_{\text{Full}} = \sum_{i=1}^{m} (L_{A_i} + \alpha L_{D_i})$$

where α is a hyperparameter to trade off the two losses and m denotes the size of minibatch samples for training the deep representation learning model.

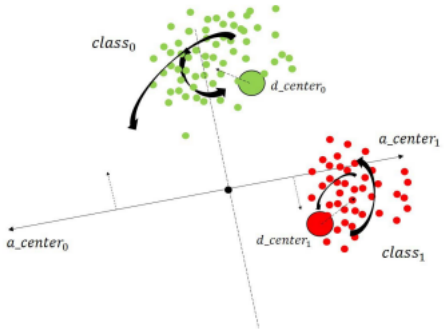

Figure 3.2: FCL composed of ACL and DCL

The detailed explanation of each element is shown in the following:

1. **Angle Center Loss (ACL)**

   It is an upgraded Softmax Loss (SL) able to improve the classification ability. Although the SL performs well in many classification tasks, it lacks effectiveness in generating distinctive features. For this reason, two constraints are introduced to keep the angular separability of instances from different classes.

   Let (W_0, b_0) and (W_1, b_1) be the weights and bias of the SL layer corresponding to class 0 and 1, respectively. The two constraint introduced are the following:

(a) • Normalization: $\|\boldsymbol{W}\| = \|\boldsymbol{W}_0\| = \|\boldsymbol{W}_1\| = 1$
• Setting of the biases to zero to maintain the angular boundary: $\|b_0\| = \|b_1\| = 0$

(b) $\boldsymbol{W}_0$ and $\boldsymbol{W}_1$ are in opposite direction: $\boldsymbol{W} = \boldsymbol{W}_0 =$ - $\boldsymbol{W}_1$

Let $f_i \in \mathbb{R}^8$ be the output of the last hidden layer of the neural network. Then, ACL can be formulated as

$$L_{A_i} = -\log\left(\frac{1}{1+e^{-2\boldsymbol{W}^T f_i}}\right) = \log\left(1+e^{-2\boldsymbol{W}^T f_i}\right)$$

2. **Distance Center Loss (DCL)**

It is the principal responsible for the separability of the learned deep representations from different classes. It can be formulated as follows:

$$L_{D_i} = \frac{1}{2}\|f_i - c_{y_i}\|_2^2$$

where c_{y_i} denotes the corresponding class center of f_i with label $y_i \in \{0, 1\}$.

It has been proven that the models supervised by DCL are trainable and can be optimized through stochastic gradient descent (SGD) method.

3.2.2 Binary Cross-Entropy Loss Function

Binary Cross-Entropy Loss Function, often abbreviated as BCE Loss or Log Loss, is a commonly used loss function in binary classification problems. It measures the difference between the actual class labels and the predicted probabilities assigned by the model for each class.
The formula for Binary Cross-Entropy Loss for a single example is:

$$L(y, \hat{y}) = -\left(y \log(\hat{y}) + (1-y)\log(1-\hat{y})\right)$$

The BCE loss function penalizes the model more when the predicted probability deviates from the actual label. Specifically:

- When the actual label is 1 (positive class), the loss function penalizes more for lower predicted probabilities ($\hat{y}$) because $log(\hat{y})$ will be closer to 0.
- When the actual label is 0 (negative class), the loss function penalizes more for higher predicted probabilities ($\hat{y}$) because $log(1-\hat{y})$ will be closer to 0.

Performance Evaluation

4.1 Metrics

Starting from the confusion matrix in Table 4.1, it is possible to compute the following evaluation metrics: Accuracy, Precision, Recall, and F1-score.

1. **Accuracy** measures the proportion of correctly classified instances (both true positives and true negatives) out of the total instances. It is computed as:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{FP} + \text{TN}}$$

2. **Precision** measures the proportion of correctly predicted positive instances (true positives) out of all instances predicted as positive. It is computed as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

3. **Recall** emphasizes the model's ability to capture all positive instances and is essential when false negatives are costly. A high Recall score indicates a low rate of false negatives, whereas a high Precision score indicates a low rate of false positives. It is given by:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

4. **F1-score** is the harmonic mean of Precision and Recall. It provides a balance between them and it is useful when there is an uneven class distribution. It is calculated as:

$$F_1 = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}}$$

	True Fraud	True Genuine
Predicted Fraud	TP	FP
Predicted Genuine	FN	TN

Table 4.1: Confusion Matrix

In addition to the aforementioned metrics, the AUC value is considered as a general performance indicator. AUC represents the relationship between the false positive rate (FP) and true positive rate (TP) at various levels. It is regarded as a superior performance metric to accuracy, as it remains unaffected by threshold selections. An AUC value nearing one means strong overall model performance.

4.2 Model Results

The best hyperparameters α, β and the learning rate μ of the optimizer are searched with the grid search method, which involves exploring a predefined set of hyperparameter combinations to identify the one that optimizes the model's performance metrics. The model is trained for each unique combination of hyperparameters and the performance of the model is then assessed using cross-validation. The hyperparameters yielding the highest performance, as measured by the accuracy, are selected as the best hyperparameters. This approach helps to fine-tune the model and improve its overall performance.

α	β	μ
0.10	0.05	0.01
0.15	0.10	0.05
0.20	0.15	0.1

Table 4.2: Grid of hyperparameter values considered for grid search. The selected values are highlighted in red.

The results of the neural network model, as presented in Table 4.3, demonstrate promising and robust performance across multiple evaluation metrics. Notably, the model achieved an accuracy of 0.982, indicating that it correctly classified 98.2% of the instances. Additionally, the recall, precision, and F1 Score values suggest that the model effectively captured the true positive instances while minimizing false positives and false negatives. The AUC value of 0.982 further underscores the model's ability to discriminate between positive and negative instances. Overall, these results indicate that the neural network model shows considerable potential for the binary classification task.

Accuracy	Recall	Precision	F1 Score	AUC
0.982	0.979	0.985	0.982	0.982

Table 4.3: Evaluation results of the neural network model on the test dataset.

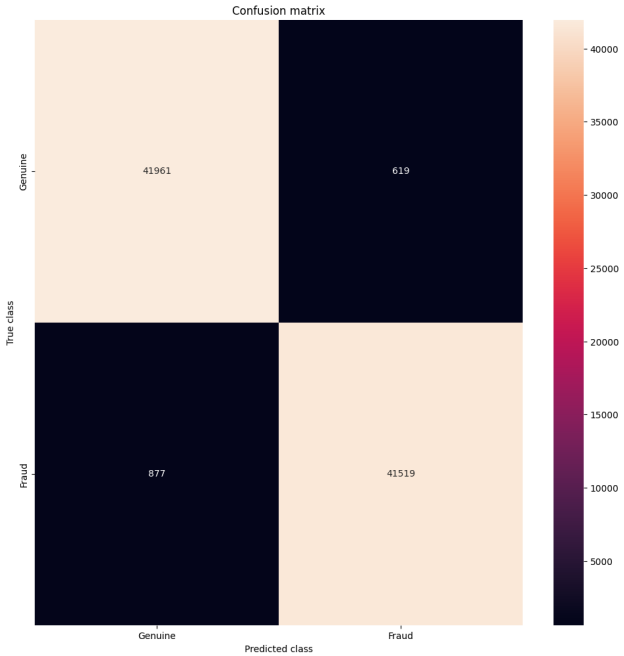

Figure 4.1: Confusion Matrix: model performance on test dataset.

Conclusions

5.1 Weaknesses Of The Paper

In the critical analysis of the paper, some areas of weakness have been identified that require attention and improvements. In particular, the following shortcomings are highlighted:

1. The preprocessing phase may be not adequately addressed.
2. The paper mentions the inclusion of the feature engineering phase but this process is not feasible due to confidentiality constraints.
3. Incorrect notation is found in different parts of the paper, such as the use of variables like a_i, b_i in the Bat Algorithm not clearly defined or used in the analysis. This inconsistency in notation could compromise the overall understanding of the work.
4. Lack of details regarding the features selected by Bat Algorithm and all the hyperparameters used.
5. The discrepancy in the models utilized for the two datasets considered in the paper to test the proposed method could potentially impact the generalizability of the findings.

5.2 Final Considerations

The article introduces a novel approach to credit card fraud detection. The integration of the Bat Optimization Algorithm addresses the challenge of feature selection in the model, allowing for the identification of relevant features. This not only enhances the model's predictive power but also streamlines the computational process, making it more efficient and scalable for real-time applications. Moreover, the adaptability of the algorithm enables it to continuously evolve and improve over time, ensuring its relevance and effectiveness in combating emerging fraud schemes and techniques.
The model demonstrates notable benefits in achieving a good performance and experimental findings highlight the efficacy of the proposed method.
Overall, it marks a significant advancement in the field of credit card fraud detection, paving the way for more accurate, reliable, and adaptive fraud prevention systems.

Bibliography

[1] Sikkandar, H., S, S., N, S., B, M. (2023). *An intelligent approach for anomaly detection in credit card data using bat optimization algorithm.* Inteligencia Artificial, 26(72), 202–222. https://doi.org/10.4114/intartif.vol26iss72pp202-222.

[2] X.-S. Yang, *A new metaheuristic bat-inspired algorithm*, in Nature Inspired Cooperative Strategies for Optimization (NICSO 2010). Berlin, Germany: Springer, 2010, pp. 65–74.

[3] Z. Li, G. Liu and C. Jiang, *Deep Representation Learning With Full Center Loss for Credit Card Fraud Detection*,in IEEE Transactions on Computational Social Systems, vol. 7, no. 2, pp. 569-579, April 2020, doi: 10.1109/TCSS.2020.2970805.

[4] Credit card fraud dataset from Kaggle: `https://www.kaggle.com/mlg-ulb/creditcardfraud`

Complete CNN Model

When considering a dataset with millions of transactions with informative feature matrices, a more powerful CNN can be adopted to learn deep representations.

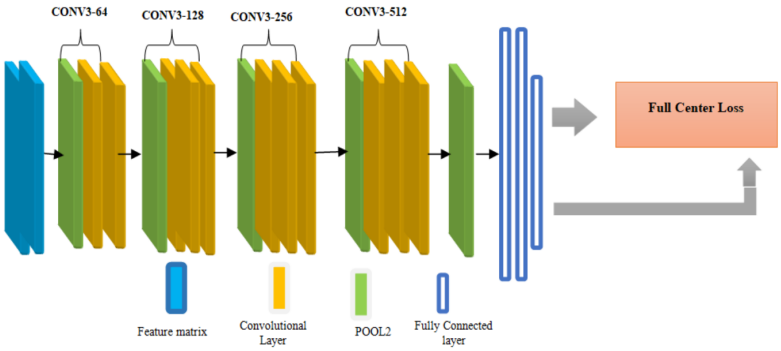

Figure A.1: Proposed architecture of Deep representation learning with FCL

CNN Layers

Convolutional layers, pooling layers, and fully connected layers constitute the three fundamental types of layers within a CNN architecture.

- **Convolution Layer**: this is the initial layer responsible for extracting distinct characteristics from the input data. Within this layer, the convolution operation occurs between the input data and a filter of size M x M. The dot product between the filter and the sections of the input data concerning to the size of the filter is taken by sliding the filter across the input dataset (M x M)
- **Pooling Layer**: the primary objective of this layer is to decrease the size of the convolved feature map, thereby reducing computational costs. This is achieved by decreasing the interconnections between layers and processing each feature map independently.
- **Fully Connected Layer**: this layer is used to link the neurons between two layers.

Moreover, in addition to these three layers, there are two additional significant factors: the dropout layer and the activation function.

- **Dropout**: it is a regularization technique to prevent the risk of overfitting and improve generalization performance. During training a fraction of the input units (or neurons) randomly selected is set to zero with a specified probability. This means that the output of those units is ignored, making the model more robust and less sensitive to noise in the training data.
- **Activation Function**: it plays a crucial role by introducing non-linear transformations, allowing the neural network model to learn complex relationships between input and output.