Neural Computing and Applications
https://doi.org/10.1007/s00521-023-09410-2

ORIGINAL ARTICLE

Federated learning model for credit card fraud detection with data balancing techniques

Mustafa Abdul Salam[1,2] · Khaled M. Fouad[1,3] · Doaa L. Elbably[1] · Salah M. Elsayed[1,4]

Received: 16 August 2023 / Accepted: 13 December 2023

Abstract

In recent years, credit card transaction fraud has resulted in massive losses for both consumers and banks. Subsequently, both cardholders and banks need a strong fraud detection system to reduce cardholder losses. Credit card fraud detection (CCFD) is an important method of fraud prevention. However, there are many challenges in developing an ideal fraud detection system for banks. First off, due to data security and privacy concerns, various banks and other financial institutions are typically not permitted to exchange their transaction datasets. These issues make traditional systems find it difficult to learn and detect fraud depictions. Therefore, this paper proposes federated learning for CCFD over different frameworks (TensorFlow federated, PyTorch). Second, there is a significant imbalance in credit card transactions across all banks, with a small percentage of fraudulent transactions outweighing the majority of valid ones. In order to demonstrate the urgent need for a comprehensive investigation of class imbalance management techniques to develop a powerful model to identify fraudulent transactions, the dataset must be balanced. In order to address the issue of class imbalance, this study also seeks to give a comparative analysis of several individual and hybrid resampling techniques. In several experimental studies, the effectiveness of various resampling techniques in combination with classification approaches has been compared. In this study, it is found that the hybrid resampling methods perform well for machine learning classification models compared to deep learning classification models. The experimental results show that the best accuracy for the Random Forest (RF); Logistic Regression; K-Nearest Neighbors (KNN); Decision Tree (DT), and Gaussian Naive Bayes (NB) classifiers are 99,99%; 94,61%; 99.96%; 99,98%, and 91,47%, respectively. The comparative results show that the RF outperforms with high performance parameters (accuracy, recall, precision and f score) better than NB; RF; DT and KNN. RF achieve the minimum loss values with all resampling techniques, and the results, when utilizing the proposed models on the entire skewed dataset, achieved preferable outcomes to the unbalanced dataset. Furthermore, the PyTorch framework achieves higher prediction accuracy for the federated learning model than the TensorFlow federated framework but with more computational time.

Keywords Credit card fraud detection (CCFD) · Federated learning · Data privacy · Class imbalance · Undersampling · Oversampling

✉ Mustafa Abdul Salam
mustafa.abdo@fci.bu.edu.eg; mustafa.abdo@aou.edu.eg

[1] Faculty of Computers and Artificial Intelligence, Benha University, Benha, Egypt

[2] Faculty of Computer Studies, Arab Open University, Cairo, Egypt

[3] Faculty of Computer Science and Engineering, New Mansoura University, Mansoura, Egypt

[4] Higher Institute for Computers & Information Technology, ElShorouk, Cairo, Egypt

Published online: 20 January 2024

1 Introduction

Credit card transactions have significantly increased in recent years due to the quick development of electronic services, including e-commerce, electronic banking, mobile payments, and the widespread use of credit cards. Without strict verification and oversight, widespread credit card uses, and many transaction situations will result in billions of dollars in losses from credit card fraud. It is challenging to calculate the loss accurately. However, according to the Nilson Report [1], Fraud losses in all other countries totaled 18.39 billion dollars in 2018. This compares to 14.99 billion dollars in 2017. Total payment card volume worldwide is expected to reach 57.080 $ trillion in 2023, with gross card fraud reaching 35.67 $ billion. This number is expected to increase significantly in the coming years. Global gross losses from card fraud will reach 40 $ billion by 2027.

Fraudulent transactions may be done using either a stolen card from internal or external sources or false information about credit cards [2]. Activities of credit card fraud detection have been widely discussed by multiple researchers [3–7]. Most of these proposed algorithms have used supervised machine learning models to recognize whether a transaction is fraudulent or legitimate. Detecting credit card fraud is an important step in stopping fraud incidents. However, there are main challenges in the development of an ideal fraud detection system for banks, such as dataset insufficiency, and skewed distribution.

Dataset insufficiency: The lack of available public datasets is the main issue associated with FDS. Data security and privacy concerns-imposed barriers to data sharing for different banks. Therefore, in this study, a federated learning approach has been deployed to allow different banks to exchange datasets to construct an efficient fraud detection model without disclosing the privacy of each bank's clients. The federated learning strategy aims to build a global integral model constructed by aggregating locally computed updates of the shared fraud detection model on distributed datasets without sharing raw data while preserving data privacy [8, 9].

Skewed distribution (class imbalance): All banks' credit card transactions are very imbalanced; just a small percentage of them involve fraud, while the majority involve legitimate purchases. In the majority of cases, 98% percent of transactions are normal, while less than 2% percent of transactions are fraudulent. In just this situation, it is particularly challenging for predictive modeling algorithms to find patterns in the data from the minority class. As a result, classifier performance is significantly impacted by skewed class distribution. The problem of class imbalance that occurred in several domains has been addressed in several ways [10–13]. Figure 1 depicts a block diagram of the FDS with an unbalanced dataset.

1.1 Motivation and contributions

The following resampling methods have been suggested as a preliminary step in processing the credit card transaction unbalanced dataset: the Oversampling techniques such as minority oversampling technique (Smote); Adaptive synthetic sampling (AdaSyn), and Random oversampling (ROS). The undersampling techniques like random undersampling (RUS).

According to previous studies, several class balance approaches have been shown to cause classification algorithms to perform with varying degrees of accuracy. This prompts us to select several classification algorithms to compare their performance after using data balancing strategies. The generated dataset is utilized for training and testing various conventional machine learning and deep learning algorithms after applying the balanced distribution of the imbalanced class using the resampling approaches outlined above. The following list represents the machine learning and deep learning algorithms used in this study: RF; DT; NB; KNN; LR, and Convolutional Neural Network (CNN).

Next, comparative research on the effect of resampling techniques on the effectiveness of classification algorithms has been conducted. Also, the appropriate techniques for handling data imbalance problems are proposed. Finally, a federated learning model over multiple frameworks to preserve the data security and privacy challenges has been built, as shown in Fig. 2.

Our contribution is summarized as.

Firstly, we applied the individual and hybrid resampling techniques with the common of machine learning classifiers, then to ensure the performance of the hybrid resampling techniques with machine learning, we compared the proposed hybrid approach with six of the state of arts.

Secondly, we applied the individual and hybrid resampling techniques with the CNN classifier, then to ensure the performance of the individual resampling techniques with CNN, we compared the proposed hybrid approach with two state-of-the-art.

Thirdly, after handling the unbalanced data, we built the federated learning model to handle the big issue of credit card fraud detection that learn the model with the training data distributed on their local database. With this approach, financial institutions can collectively reap the benefits of a shared global model, which has seen more fraud than each bank alone, without sharing the dataset.

Finally, we executed the proposed federated learning model with different optimization techniques and with several batch sizes over different platforms (Pytorch and

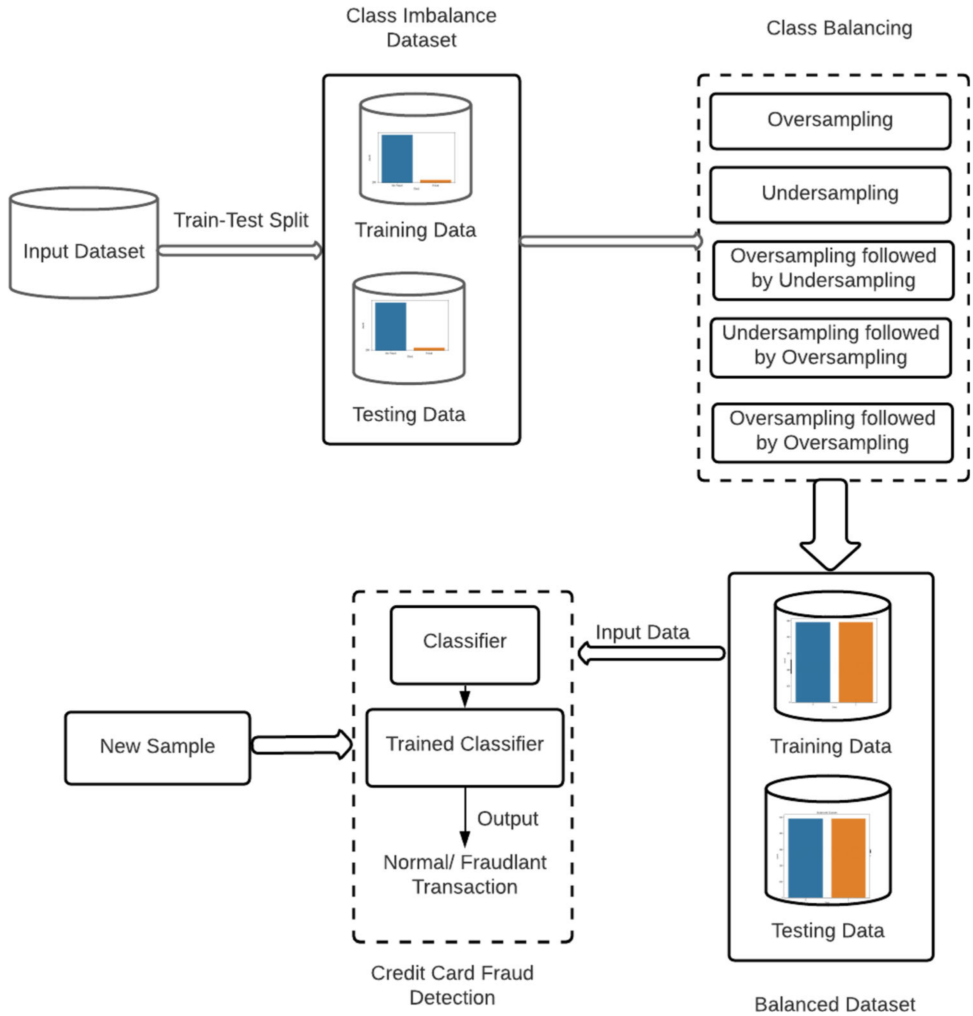


Fig. 1 Block diagram of CCFD model with an unbalanced dataset

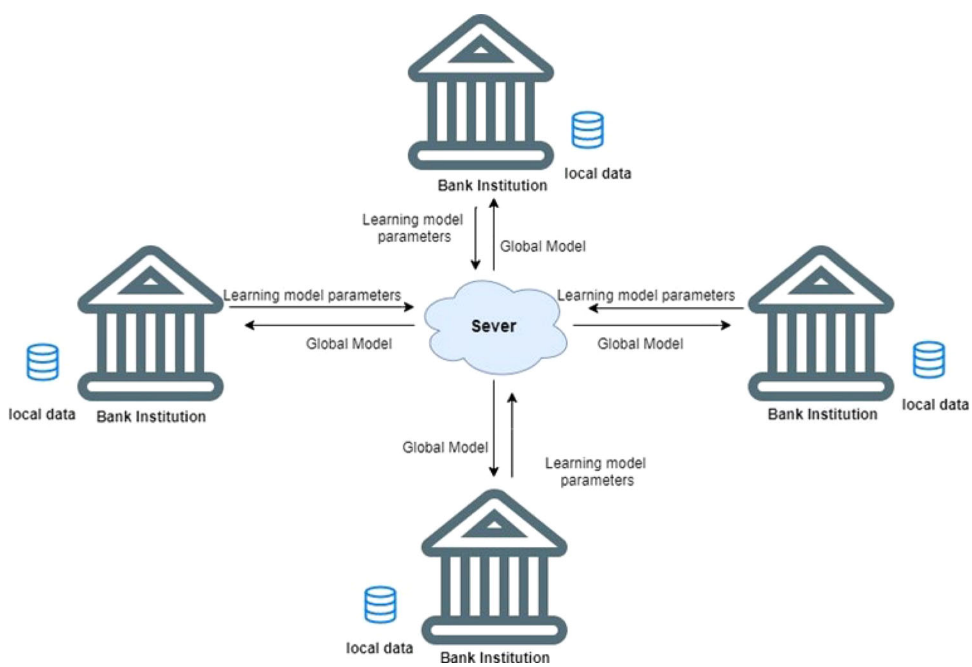


Fig. 2 Federated learning model for FDS

Tensorflow federated) to get the best platform according to accuracy and computation time.

The following sections of this paper are presented as follows: In Sect. 2 is the review of all previous works of federated learning models to identify the fraudulent transactions, imbalance classification problem of CCFD and the integration between them. Section 3 shows a background of all used resampling methods. In Sect. 4, the details of the proposed hybrid resampling approaches with pseudocode of each approach. In Sect. 5, the main steps of the proposed federated learning model. The experimental results of all proposed hybrid resampling approaches using machine learning classifications and CNN classifier on benchmark dataset and the effectiveness of the federated learning model on different platforms were explained in Sect. 5. Finally, in Sect. 6 this paper is concluded and briefly suggestion our future works.

2 Related work

Fraud detection algorithms use machine learning to efficiently identify fraudulent transactions. Most proposed CCFDS are built using centralized learning models, and a handful of researchers are building federated learning models to tackle fraud detection. The supervised, unsupervised, and semi-supervised learning models use centralized learning strategies [14–18]. Fraud detection is viewed as a classification issue for a set of card transactions in data mining tasks. A comparison study on CCFD [19] has been by using supervised approaches such as Extreme Gradient Boosting (XGB); DT; RF; LR; K-NN, and SVM and unsupervised approaches such as Generative Adversarial Networks (GAN); Auto-Encoder (AE), Restricted Boltzmann Machine (RBM), and One-Class SVM (OCSVM). The authors [20] evaluated the performances of various ML techniques like SVM; KNN; DT, and NB for CCFD.

The Federated learning (FL) concept has an important role in the banking industry, especially in the fraud detection of credit cards. With the development of credit card fraud detection systems, there is a problem of data security and privacy protection, and FL will solve this problem [21]. This paper proposed a federated Neural Network Model. As a proper deep-learning model for identifying credit card fraud. However, it is unconcerned about the issue of privacy. In [22], this work applies a federated learning model for detecting Credit card fraudulence. This paper evaluates CCFD with a federated learning model for a real-time dataset. Compared to centralized deep learning models, this increases by the AUC test average of 10%.

In [23], the authors proposed two unsupervised deep learning models (AE; RBM) to identify credit card fraud using only a small number of parameters. For AE and RBM, the accuracy rate of federated deep learning models is 88% and 94% percent, respectively, while for centralized deep learning models, it is 99%and 92%percent [24]. This work introduces a new protocol that is an efficient and privacy-preserving strategy based on FL with a stochastic gradient descent method by combining differential privacy with homomorphic encryption. The authors [25] surveyed various types, including behavioral fraud, application fraud, counterfeit fraud, theft fraud, and bankruptcy fraud. Furthermore, the performance metrics for fraudulence are predicted by a decision tree, clustering algorithms, pairwise matching, neural network, and genetic algorithms. In order to combine the features in local and global models and obtain high performance with minimal communication expense, a feature fusion technique is developed [26].

Ref. [24] This work introduces a new protocol that is an efficient and privacy-preserving strategy based on FL with a stochastic gradient descent method by combining differential privacy with homomorphic encryption. The authors [25] surveyed various types, including behavioral fraud, application fraud, counterfeit fraud, theft fraud, and bankruptcy fraud. Furthermore, the performance metrics for fraudulence are predicted by a decision tree, Suvasini, et al. [27] presented comparative research on credit card fraud detection utilizing seven widely used classification

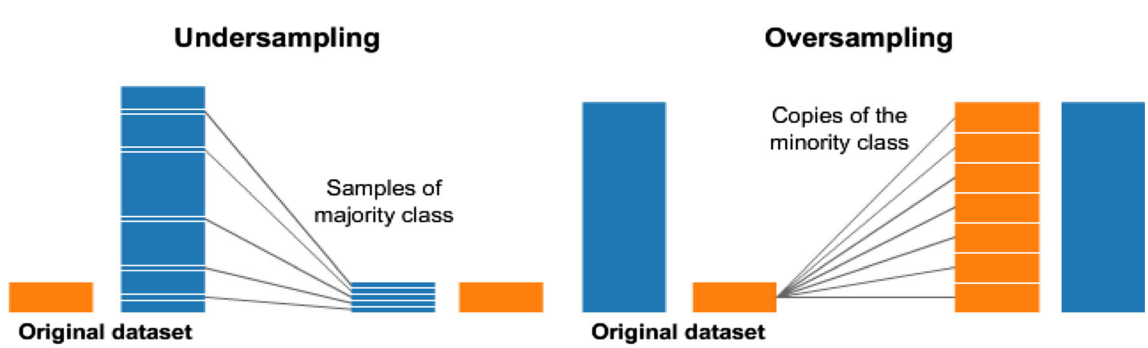


Fig. 3 Main resampling Techniques [38]

techniques. The experimental findings demonstrated that, for real-time datasets, the decision tree classifier outperforms the other classifiers at predicting credit card fraud. However, the SVM model still detects fewer fraudulent transactions than the decision tree model does. Mohd [28] developed the genetic algorithm and scatter search techniques. The credit card limit is assumed and used as the cost of misclassification. This proposed technique determines the credit card's available limit based on fraudsters' use of this available limit. Kundu et al. [29] suggested a method to understand the transaction sequence by using the model of Hidden Markov and the K-Clustering Model. Based on the cardholder's spending behavior, the proposed model created clusters for low, medium, and high spending amounts. It has been demonstrated that the model's outcomes speed up fraudulence detection.

The class imbalance problem, which has drawn considerable interest from the various application fields of machine learning-based classification approaches, is the primary obstacle to developing a prediction model for CCFD [30, 31]. To address the unbalanced data, many approaches have been developed in different domains. Huang et al. [32] used deep learning to handle the imbalanced data in face analysis. They have handled this problem using a cost-sensitive approach and class resampling technique. In [33], Ouyang et al. proposed a framework for oil spill problems. The authors have proved that the imbalanced dataset problem decreases the learning model's performance. Yang et al. [34] introduced a Sample Subset Optimization technique that handles the class imbalance distribution problem in Bioinformatics applications using ensemble learning. Sun et al. [35] created the EUS-Bag fitness function, an evolutionary under-sampling method based on a bagging ensemble framework. The PSOAANN approach, a hybrid of Particle Swarm Optimization and Auto-Associative Neural networks, was proposed by Kamaruddin and Ravi [36]. Wei et al. [37] presented an efficient solution to the unbalanced data issues for online credit card fraud detection.

3 Materials and methods

3.1 Resampling techniques

The resampling approach is a popular method for handling incredibly imbalanced datasets. Resampling approaches come in two varieties: the undersampling technique removes certain samples from the majority class (blue color data) the oversampling technique adds more examples from the minority class (orange color data), as demonstrated in Fig. 3.

3.1.1 Oversampling techniques

3.1.1.1 Random oversampling technique (ROS) To address the issue of class imbalance, ROS [39] is a useful and widely used oversampling method. ROS methodology: duplicate samples from minority classes that are chosen at random. Then, while training the machine learning models, combine this new sample with the original data. The original minority dataset is partially recreated using this random oversampling technique, which increases the likelihood that the model will overfit.

3.1.1.2 Synthetic minority oversampling technique (smote) Smote [40] is common for class imbalance problems. This resampling technique uses synthetic data points existing created by interpolating new instances between available data points of the minority class. The K-Nearest Neighbors (KNN) algorithm is used to construct the interpolation of the instances of synthetic data. The KNN selects new minority class data points according to the requirements of synthetic data instances, and then adds them to the original dataset. The Smote technique will perform efficiently. In this case, the size of the datasets is small. But, when the size of datasets is large. The process will not function effectively, and creating more synthetic data points will need more calculation time.

3.1.1.3 Adaptive synthetic sampling (AdaSyn) Adaptive oversampling is a technique used in [41]. It is suggested to avoid the limitations of Smote technique. These limitations occur while creating the synthetic data samples. Smote technique may make it more likely that data points may overlap. In AdaSyn oversampling approach, the synthetic data instances are produced using the density distribution of the minority class. AdaSyn enhances the imbalanced dataset by rebalancing it and reducing the learning bias.

3.1.2 Undersampling techniques

3.1.2.1 Random undersampling (RUS) RUS is the most popular and effective resampling technique for class-imbalanced datasets [42]. Although the RUS is quicker than other resampling methods, it causes losing-out valuable data. Therefore, the RUS method decreases the performance of the classification algorithm while learning.

4 The proposed hybrid resampling techniques

4.1 Oversampling followed by undersampling

4.1.1 ROS followed by RUS

To balance the class distribution of a dataset, this approach employs a hybrid resampling strategy that combines random oversampling (ROS) and random undersampling (RUS). The algorithm is fed four parameters: X, P_{RUS}, P_{ROS}, and Nmin. X is the original dataset, which includes samples from both the majority and minority classes. P_{RUS} is the proportion of RUS that defines how many samples from the majority class are deleted. The percentage of ROS that decides how many minority class samples will be replicated is known as P_{ROS}. The number of minority class samples in X is denoted by N_{min}.

The algorithm is divided into two steps: oversampling and undersampling. The technique creates N_{ROS} new minority class samples in the oversampling stage by randomly picking and duplicating N_{min} samples. By multiplying N_{min} by P_{ROS}, N_{ROS} is calculated. The additional samples are saved in an array S_R before being added to X to create a new dataset S_{ROS}. The approach removes N_{RUS} majority class samples from S_{ROS} during the undersampling step by randomly picking and deleting N_{maj} samples. By increasing N_{maj} by P_{RUS}, we get N_{RUS}. In S_{ROS}, N_{maj} is the number of majority class samples. The array S $_{(ROS+\ RUS)}$, which is the algorithm's final output.

Algorithm 1 ROS + RUS

Input: - X is the original dataset
P_{RUS} percent of RUS
P_{ROS} percent of ROS
N_{min} the number of minority class in X
Output: - The resampled dataset **S** $_{\mathbf{(ROS+\ RUS)}}$
$N_{ROS}= N_{min} * P_{ROS}$
For i=1 to N_{ROS}
a- Choose randomly a sample from $\mathbf{N}_{min}$
b- Duplicates sample and save it to new array $\mathbf{S_R}$
end
$\mathbf{S_{ROS}} = = \boldsymbol{X} \cup \boldsymbol{S_R}$
N_{maj} the number of majority class in $\mathbf{S_{ROS}}$
$\mathbf{N_{RUS}} = \mathbf{N}_{maj} * P_{RUS}$
For i=1 to N_{RUS}
a- Choose randomly a sample from $\mathbf{N}_{maj}$ and call it N_s
b- Delete sample N_s and save it to new array $\mathbf{S_{(ROS+RUS)}}$
end

4.1.2 Smote followed by RUS

To deal with imbalanced data, this algorithm is a hybrid sampling technique that combines oversampling (SMOTE) and undersampling (RUS) methods. The algorithm aims to improve classification model accuracy by generating a more representative dataset for both classes. It employs SMOTE to increase the diversity and density of the minority class, and RUS to reduce the majority class's noise and redundancy. The algorithm creates a new dataset called SSmote, which contains more instances of the minority class than the previous dataset, X, by combining it with the synthetic set S. After that, Ns = Nmaj * PR instances from the majority class in SSmote are chosen at random and removed from the dataset. These instances are saved to the final resampled dataset, a new array S(Smote + RUS). The output of the algorithm is S(Smote + RUS), which has a more evenly distributed class distribution than X.

Algorithm 2 Smote + RUS

Input: - X is the original dataset
K the number of the nearest neighbors
PR percent of RUS
PS percent of Smote
Nmin the number of minority class
Output: - The resampled dataset S(Smote+RUS)
For i=1 to N_{min} **do**
1-Find the k nearest (minority class) neighbors of X_i
2- $\hat{N} = [P_S/100]$
While $\hat{N} \neq 0$ **do**
a- Select one of the K's nearest neighbors and call this $\bar{X}$
b- Select a random number $\alpha \in [0,1]$
c- $\hat{X} = X_i + \alpha\,(\bar{X} - X_i)$
d- Append $\hat{X}$ to S
e- $\hat{N} = \hat{N}$-1
end
end
$\mathbf{S_{Smote}} = \boldsymbol{X} \cup \boldsymbol{S}$
N_{maj} the number of majority class in $\mathbf{S_{Smote}}$
$\mathbf{N_s} = \mathbf{N}_{maj} * P_R$
For i=1 to N_s
a- Choose randomly a sample from $\mathbf{N}_{maj}$ and call it N_s
b- Delete sample N_s and save it to a new array $\mathbf{S_{(Smote+RUS)}}$
end

4.1.3 AdaSyn followed by RUS

To deal with imbalanced data, is a hybrid sampling technique that combines the AdaSyn and RUS algorithms. The pseudocode begins by calculating the number of synthetic data samples required for the minority class. Then, for each instance of the minority class Xi, it locates its K nearest neighbors of the same class and computes a ratio Ri that measures how difficult it is to learn Xi based on how many of its neighbors are members of the majority class. First, applying AdaSyn that creates the synthetic data instances using the by selecting one of the K neighbors Xsi at random and interpolating between Xi and Xsi with a uniform random factor, then combines with the original dataset X to form the new dataset S_{AdaSyn}, which contains more instances of the minority class than before. Following that, using RUS, it selects Ns = Nm * P instances at random from the majority class in S_{AdaSyn} and deletes them from the dataset.

Algorithm 3 AdaSyn + RUS

Input: - X is the original dataset

β a specified parameter to a desired balanced level

K the number of the nearest neighbors

P percent of RUS

Output: - The resampled dataset $S_{(AdaSyn+ RUS)}$

$\mathbf{S_{maj}}$ Majority Class, $\mathbf{S_{min}}$ Minority Class

$\mathbf{N_{maj}}$ Number of majority observations in X, $\mathbf{N_{min}}$ Number of minority observations in X

1. Calculate the number of synthetic data samples that need to be generated for the minority class as follows

$$G = (N_{maj} - N_{min}) * \beta \qquad [50]$$

2. Find the K-nearest neighbors based on the Euclidean distance and store the indices in nn

For i=1 to N_{min} **do**

$$\boldsymbol{R_i = \frac{(nn_i \cap S_{maj})}{K}} \qquad \textbf{[51]}$$

3. Normalize Ri using $\hat{R}_{new(i)} = \frac{R_i}{\sum_{i=1}^{N_{maj}} R_i}$

4. Calculate the number of synthetic data samples that need to be generated for the minority sample $X_{(i)}$

$$g_i = \hat{R}_{new(i)} * G \qquad [51]$$

5. Generate the synthetic data samples g_i for each minority class data sample $X_{(i)}$

For i=1 to g_i **do**

a- Randomly choose one minority data sample $X_{(si)}$ from the k-nearest neighbor for data $X_{(i)}$

b- Generate the synthetic data sample as follows:

$$S_i = X_{(i)} + (X_{(si)} - X_{(i)}) * unifrom(0,1) \qquad [40]$$

end

end

$S_{AdaSyn}=(S \cup X)$

N_m the number of the majority class in S_{AdaSyn}

$\mathbf{N_s=N_m*P}$

For i=1 to N_s

a- Choose randomly a sample from N_m and call it N_s

b- Delete N_s and save it to a new array $S_{(RUS)}$

end

$\mathbf{S_{(AdaSyn+ RUS)}} = (S_{AdaSyn} \cup S_{RUS})$

4.2 Undersampling followed by oversampling

4.2.1 RUS followed by ROS

This hybrid between undersampling techniques (RUS) then Oversampling technique (ROS). First, the number of instances to be deleted from the majority class is calculated as $N_{RUS} = N_{maj} * P_{RUS}$. It chooses N_{RUS} instances at random from the majority class in X and deletes them from the dataset. It saves these deleted instances to a new array SRUS, which contains fewer instances of the majority class than previously. Then, the number of instances to be duplicated from the minority class is calculated as $N_{ROS} = N_{min} * P_{ROS}$. It chooses N_{ROS} instances at random from the minority class in SRUS and duplicates them in the dataset. It saves the duplicated instances to a new array SROS, which contains more instances of the minority class than previously. Finally, it combines S_{RUS} and SROS to create the final resampled dataset $S_{(RUS+\ ROS)}$.

Algorithm 4 RUS + ROS

Input: - X is the original dataset
P_{RUS} percent of RUS
P_{ROS} percent of ROS
N_{maj} the number of majority class in X
Output: - The resampled dataset $\mathbf{S}_{(RUS+\ ROS)}$
$\mathbf{N_{RUS}} = \mathbf{N}_{maj} * P_{RUS}$
For i=1 to N_{RUS}
a- Choose randomly a sample from $\mathbf{N}_{maj}$ and call it N_s
b- Delete sample N_s and save it to a new array $\mathbf{S_{RUS}}$
end
N_{min} the number of minority class in $\mathbf{S_{RUS}}$
$N_{ROS} = N_{min} * P_{ROS}$
For i=1 to N_{ROS}
a- Choose randomly a sample from $\mathbf{N}_{min}$
b-Duplicates sample and save it to new array $\mathbf{S_{ROS}}$
end

$\mathbf{S}_{(RUS+\ ROS)} = \boldsymbol{S_{RUS}} \cup \boldsymbol{S_{ROS}}$

4.2.2 RUS followed by smote

Is a hybrid method for dealing with imbalanced datasets in machine learning. To balance the class distribution, it combines random under-sampling (RUS) and synthetic minority over-sampling technique (SMOTE). First, RUS: To begin, the algorithm reduces the size of the majority class by a percentage determined by PR. It creates a subset $\mathbf{S_{RUS}}$ by randomly selecting samples from the majority class until the number of samples reaches Ns, which is calculated as a percentage of the original majority class size. Then (SMOTE): For each minority class sample in SRUS, the algorithm finds the minority class's K nearest neighbors. Then, by interpolating between the minority sample and its neighbors, it generates new synthetic samples. PS percent of SMOTE determines the number of new samples to be created. Bringing RUS and SMOTE together: Combining the under-sampled majority class dataset $\mathbf{S_{RUS}}$ with the over-sampled minority class dataset S_{Smote} yields the final resampled dataset $\mathbf{S_{(RUS+Smote)}}$.

Algorithm 5 RUS + Smote

Input: - X is the original dataset
K the number of the nearest neighbors
P_R percent of RUS
P_S percent of Smote
N_{min} the number of minority class
N_{maj} the number of majority class
Output: - The resampled dataset $\mathbf{S_{(RUS+Smote)}}$
$\mathbf{N_s} = \mathbf{N}_{maj} * P_R$
For i=1 to N_s
a- Choose randomly a sample from $\mathbf{N}_{maj}$ and call it N_s
b-Delete sample N_s and save it to a new array $\mathbf{S_{RUS}}$
end
For i=1 to N_{min} **do**
1-Find the k nearest(minority class) neighbors of $\boldsymbol{S_{RUS(i)}}$
2- $\hat{N} = [P_S/100]$
While $\hat{N} \neq 0$ **do**
a- Select one of the K's nearest neighbors and call this $\bar{X}$
b- Select a random number $\alpha \in [0,1]$
c- $\hat{X} = S_{RUS(i)} + \alpha\,(\bar{X} - S_{RUS(i)})$
d- Append $\hat{X}$ to S_{Smote}
e- $\hat{N} = \hat{N}$-1
end
end

$\mathbf{S_{(RUS+Smote)}} = \mathbf{S_{RUS}} \cup \mathbf{S_{Smote}}$

4.2.3 RUS followed by AdaSyn

Another hybrid method for dealing with imbalanced datasets in machine learning, is Algorithm 6. To balance the class distribution, it employs a combination of Random Under-Sampling (RUS) and Adaptive Synthetic Sampling (AdaSyn).

RUS: The algorithm begins by reducing the size of the majority class by a percentage defined by P. It creates a subset SRUS by randomly selecting samples from the majority class until the number of samples reaches Ns, which is calculated as a percentage of the original majority class size.

(AdaSyn): Based on a given parameter β, which represents the intended balanced level, the algorithm determines how many synthetic samples must be created for the minority class. The algorithm locates K nearest neighbors for each minority class sample in S_{RUS} and calculates a ratio Ri, which indicates how many of the neighbors are members of the majority class. After normalization, the ratio Ri is used to calculate the number of synthetic samples needed for each minority sample. The minority sample and one of its randomly selected neighbors from the minority class are interpolated to create the synthetic samples. Integrating AdaSyn and RUS: The under-sampled majority class dataset S_{RUS} and the over-sampled minority class dataset AdaSyn are combined to create the final resampled dataset $\mathbf{S}_{(\mathbf{RUS}+\text{ AdaSyn})}$.

Algorithm 6 RUS + AdaSyn

Input: - X is the original dataset

β a specified parameter to a desired balanced level

K the number of the nearest neighbors

P percent of RUS

N_m the number of the majority class

Output: - The resampled dataset $\mathbf{S}_{(\mathbf{RUS}+\text{ AdaSyn})}$

$\mathbf{N_s=N_m*P}$

For i=1 to N_s

a- Choose randomly a sample from N_m and call it N_s

b- Delete N_s and save it to a new array S_{RUS}

end

$\mathbf{S_{maj}}$ Majority Class, $\mathbf{S_{min}}$ Minority Class

$\mathbf{N_{maj}}$ Number of majority observations in S_{RUS}, $\mathbf{N_{min}}$ Number of minority observations in S_{RUS}

1. Calculate the number of synthetic data samples that need to be generated for the minority class as follows:

$$G = \left(N_{maj} - N_{min}\right) * \beta \qquad [50]$$

2. Find the K-nearest neighbors based on the Euclidean distance and store the indices in nn

For i=1 to N_{min} **do**

$$\boldsymbol{R_i = \frac{(nn_i \cap S_{maj})}{K}} \qquad \textbf{[51]}$$

3. Normalize Ri using $\hat{R}_{new(i)_i} = \frac{R_i}{\sum_{i=1}^{N_{maj}} R_i}$

4. Calculate the number of synthetic data samples that need to be generated for the minority sample $S_{RUS(i)}$

$$g_i = \hat{R}_{new(i)_i} * G \qquad [51]$$

5. Generate the synthetic data samples g_i for each minority class data sample $S_{RUS(i)}$

For i=1 to g_i do

a- Randomly choose one minority data sample X_i from the k-nearest neighbor for data $S_{RUS(i)}$

b- Generate the synthetic data sample as follows:

$$S_i = S_{RUS(i)} + \left(X_i - S_{RUS(i)}\right) * unifrom(0,1) \qquad [40]$$

end

end

$\mathbf{S}_{(\mathbf{RUS+AdaSyn})}\mathbf{=(}S \cup S_{RUS}\mathbf{)}$

4.3 Oversampling followed by oversampling

4.3.1 ROS followed by smote

This approach helps to improve classifier performance on imbalanced datasets by increasing the diversity and representation of the minority class. To balance the class distribution, it combines Random Over-Sampling (ROS) and Synthetic Minority Over-Sampling Technique (SMOTE). ROS: The algorithm begins by increasing the size of the minority class by a percentage determined by P_{ROS}. It draws samples from the minority class at random and duplicates them to form a subset SR until the number of samples reaches N_{ROS}, which is calculated as a percentage of the original minority class size.

(SMOTE): The algorithm finds K nearest neighbors from the minority class for each minority class sample in $\mathbf{S_{ROS}}$, which is the union of the original dataset X and the over-sampled subset SR. Then, by interpolating between the minority sample and its neighbors, it generates new synthetic samples. PS percent of SMOTE determines the number of new samples to be created. Combining the over-sampled minority class dataset $\mathbf{S_{ROS}}$ with the over-sampled minority class dataset S_{Smote} yields the final resampled dataset $\mathbf{S_{(ROS+\ Smote)}}$.

Algorithm 7 ROS + Smote

Input: - X is the original dataset
K the number of the nearest neighbors
P_S percent of Smote
P_{ROS} percent of ROS
N_m the number of minority class in X
Output: - The resampled dataset $\mathbf{S_{(ROS+\ Smote)}}$
$N_{ROS}= N_m{*}\ P_{ROS}$
For i=1 to N_{ROS}
a- Choose randomly a sample from $\mathbf{N_m}$
b- Duplicates sample and save it to new array $\mathbf{S_R}$
end
$\mathbf{S_{ROS}} = = \boldsymbol{X} \cup \boldsymbol{S_R}$
N_{maj} the number of majority class in $\mathbf{S_{ROS}}$
N_{min} the number of minority class in $\mathbf{S_{ROS}}$
For i=1 to N_{min} **do**
1-Find the k nearest (minority class) neighbors of $\boldsymbol{S_{ROS(i)}}$
2- $\widehat{N} = [P_S/100]$
While $\widehat{N} \neq 0$ **do**
a- Select one of the K's nearest neighbors and call this $\overline{X}$
b- Select a random number $\alpha \in [0,1]$
c- $\hat{X} = S_{ROS(i)} + \alpha\ (\overline{X} - S_{ROS(i)})$
d- Append $\hat{X}$ to S_{Smote}
e- $\widehat{N} = \widehat{N}$-1
end
end

$\mathbf{S_{(ROS+Smote)}} = \mathbf{S_{ROS}} \cup \mathbf{S_{Smote}}$

4.3.2 ROS followed by AdaSyn

The following algorithm implements the ROS + AdaSyn combination, It combines Random Over-Sampling (ROS) and Adaptive Synthetic Sampling (AdaSyn) to balance the class distribution. First, increasing the minority class size by a percentage determined by P_{ROS}. It draws samples from the minority class at random and duplicates them to form a subset SR until the number of samples reaches N_{ROS}, which is calculated as a percentage of the original minority class size. Then using AdaSyn, based on a given parameter β, which represents the intended balanced level, the algorithm determines how many synthetic samples must be created for the minority class. The method locates K nearest neighbors for each minority class sample in S_{ROS}, which is the union of the original dataset X and the over-sampled subset SR. It then computes a ratio Ri, which indicates the proportion of neighbors that are members of the majority class. After normalization, the ratio Ri is used to calculate the number of synthetic samples needed for each minority sample. The minority sample and one of its randomly selected neighbors from the minority class are interpolated to create the synthetic samples. to create the synthetic data instances based on the minority class density distribution.

Combining ROS and AdaSyn: The over-sampled minority class dataset S, which includes both duplicated and synthetic samples, is combined with the original dataset X to create the final resampled dataset $\mathbf{S_{(ROS+\ AdaSyn)}}$.

Algorithm 8 ROS + AdaSyn

Input: - X is the original dataset
K the number of the nearest neighbors
P_S percent of Smote
P_{ROS} percent of ROS
N_m the number of minority class in X
β a specified parameter to a desired balanced level
Output: - The resampled dataset $\mathbf{S_{(ROS+\ AdaSyn)}}$
$N_{ROS}= N_m * P_{ROS}$
For i=1 to N_{ROS}
a- Choose randomly a sample from $\mathbf{N}_m$
b- Duplicates sample and save it to new array $\mathbf{S_R}$
end
$\mathbf{S_{ROS}}= \boldsymbol{X} \cup \boldsymbol{S_R}$
$\mathbf{S_{maj}}$ Majority Class, $\mathbf{S_{min}}$ Minority Class
$\mathbf{N_{maj}}$ Number of majority observations in $\mathbf{S_{ROS}}$, $\mathbf{N_{min}}$ Number of minority observations in $\mathbf{S_{ROS}}$
1. Calculate the number of synthetic data samples that need to be generated for the minority class as follows:
$$G = \left(N_{maj} - N_{min}\right) * \beta \qquad [50]$$
2. Find the K-nearest neighbors based on the Euclidean distance and store the indices in nn

For i=1 to N_{min} **do**
$$\boldsymbol{R_i = \frac{(nn_i \cap S_{maj})}{K}} \qquad \textbf{[51]}$$
3. Normalize Ri using $\hat{R}_{new(i)_i} = \frac{R_i}{\sum_{i=1}^{N_{maj}} R_i}$
4. Calculate the number of synthetic data samples that need to be generated for the minority sample $S_{ROS(i)}$
$$g_i = \hat{R}_{new(i)} * G \qquad [51]$$
5. Generate the synthetic data samples g_i for each minority class data sample S_{ROS}

For i=1 to g_i do
a- Randomly choose one minority data sample X_i from the k-nearest neighbor for data $S_{ROS(i)}$
b- Generate the synthetic data sample as follows:
$$S_i = S_{ROS(i)} + \left(X_i - S_{ROS(i)}\right) * unifrom(0,1) \qquad [40]$$
end
end

$\mathbf{S_{(ROS+AdaSyn)}} = (S \cup S_{ROS})$

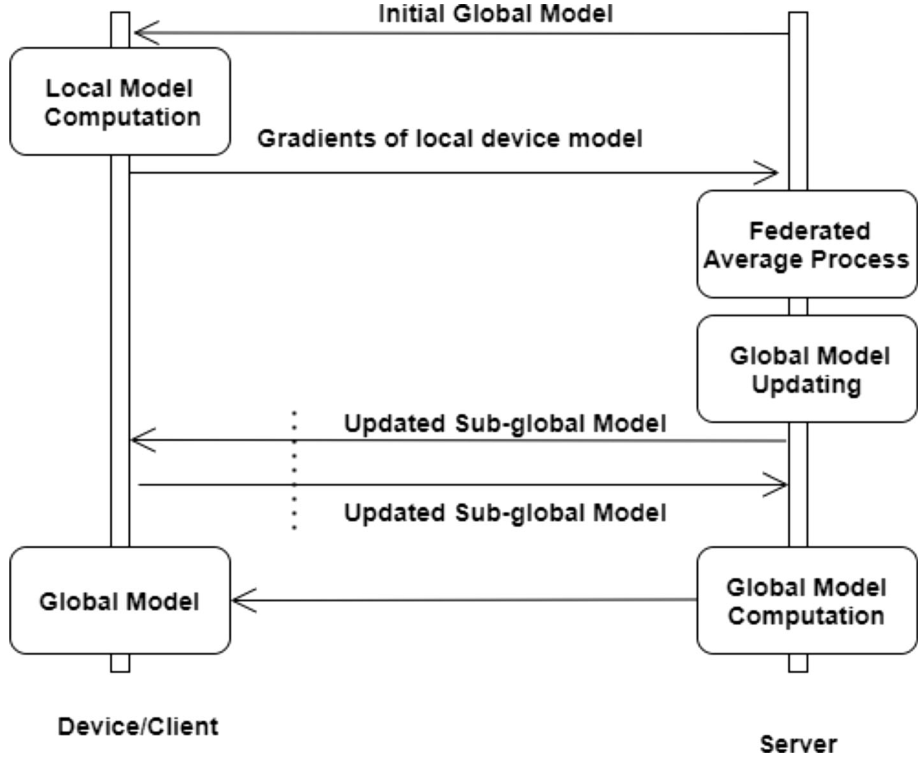


Fig. 4 The main steps of the client–server process

4.3.3 Smote followed by AdaSyn

Is a hybrid method for dealing with imbalanced datasets in machine learning. The following description for each phase:

4.3.3.1 Synthetic minority over-sampling technique (SMOTE) The algorithm determines K nearest neighbors from the minority class for each minority class sample in the original dataset X. Next, by interpolating between the minority sample and its neighbors, it creates new synthetic samples. PS percent of SMOTE determines how many new samples need to be created. SSmote is created by combining the original dataset X with the oversampled minority class dataset S.

4.3.3.2 Adaptive synthetic sampling (AdaSyn) Based on a given parameter {\, which represents the intended balanced level, the algorithm determines how many synthetic samples must be generated for the minority class. The algorithm locates K nearest neighbors for each minority class sample in SSmote and calculates a ratio Ri, which indicates how many of the neighbors are members of the majority class. After normalization, the ratio Ri is used to calculate the number of synthetic samples needed for each minority sample. The minority sample and one of its randomly selected neighbors from the minority class are interpolated to create the synthetic samples.

Table 1 Overview of the dataset obtained from Kaggle

Total dataset	#Fraud	#Not fraud	Label not fraud	Label fraud
284,807	492	284,315	0	1

4.3.3.3 Combining SMOTE and AdaSyn The over-sampled minority class dataset S, which is made up of both synthetic samples produced by SMOTE and AdaSyn, is combined with the original dataset X to create the final resampled dataset $\mathbf{S}_{(\mathbf{Smote+AdaSyn})}$.

Algorithm 9 Smote + AdaSyn

Input: - X is the original dataset
P_S percent of Smote
N_{min} the number of minority class
N_{maj} the number of majority class
β a specified parameter to a desired balanced level
K the number of the nearest neighbors
Output: - The resampled dataset $\mathbf{S}_{(Smote+AdaSyn)}$
For i=1 to N_{min} **do**
1-Find the k nearest (minority class) neighbors of X_i
2- $\hat{N} = [P_S/100]$
While $\hat{N} \neq 0$ **do**
a- Select one of the K's nearest neighbors and call this $\bar{X}$
b- Select a random number $\alpha \in [0, 1]$
c- $\hat{X} = X_i + \alpha(\bar{X} - X_i)$
d- Append $\hat{X}$ **to S**
e- $\hat{N} = \hat{N}-1$
end
end
$\mathbf{S}_{Smote} = X \cup S$
$\mathbf{S}_{maj}$ Majority Class, $\mathbf{S}_{min}$ Minority Class
$\mathbf{N}_{maj}$ Number of majority observations in $\mathbf{S}_{mote}$, $\mathbf{N}_{min}$ Number of minority observations in $\mathbf{S}_{Smote}$
3. Calculate the number of synthetic data samples that need to be generated for the minority class as follows:
$$G = (N_{maj} - N_{min}) * \beta \qquad [50]$$
4. Find the K-nearest neighbors based on the Euclidean distance and store the indices in nn
For i=1 to N_{min} **do**
$$R_i = \frac{(nn_i \cap S_{maj})}{K} \qquad [51]$$
5. Normalize Ri using $\hat{R}_{new(i)} = \frac{R_i}{\sum_{i=1}^{N_{maj}} R_i}$
6. Calculate the number of synthetic data samples that need to be generated for the minority sample $S_{Smote(i)}$
$$g_i = \hat{R}_{new(i)} * G \qquad [51]$$
7. Generate the synthetic data samples g_i for each minority class data sample S_{Smote}
For i=1 to g_i do
a- Randomly choose one minority data sample X_i from the k-nearest neighbor for data $S_{Smote(i)}$
b- Generate the synthetic data sample as follows:
$$S_i = S_{Smote(i)} + (X_i - S_{Smote(i)}) * unifrom(0,1) \qquad [40]$$
end
end
$\mathbf{S}_{(Smote+AdaSyn)} = (S \cup S_{Smote})$

Table 2 Comparison results of common machine learning classifiers after applying resampling techniques on credit card dataset

Resampling Method	Model	Accuracy	Precision	Recall	F1 Score	Loss	Computation Time (Second)
ROS	Random forest	**99.99**	**0.9999**	**1.0**	**0.9999**	**0.0016**	**1090**
	Logistic regression	94.57	97.44	0.9155	0.9440	1.872	32
	KNN	99.96	0.9993	1.0	0.9996	0.0111	754
	Decision tree	99.98	0.9996	1.0	0.9998	0.0068	**<u>84</u>**
	GaussianNB	91.48	0.9708	0.8553	0.9094	2.940	8
Smote	Random forest	**99.98**	0.9997	1.0	**0.9998**	**0.0036**	**2240**
	Logistic regression	94.51	0.9730	0.9156	0.9434	1.895	34
	KNN	**99.91**	0.9983	1.0	**0.9991**	0.0289	**814**
	Decision tree	99.82	0.9975	0.9989	0.9982	0.0597	270
	GaussianNB	91.43	0.9723	0.8530	0.9087	2.956	6
AdaSyn	Random forest	**99.98**	09997	1.0	**0.9998**	0.0039	**2683**
	Logistic regression	89.78	0.9076	0.8857	0.8966	3.528	40
	KNN	99.91	0.9982	1.0	0.9991	0.0295	821
	Decision tree	99.86	0.9979	0.9994	0.9986	0.0452	345
	GaussianNB	73.32	0.9267	0.5064	0.6549	9.214	10
RUS	Random forest	93.91	0.9763	0.8993	0.9355	2.102	6
	Logistic regression	94.34	0.9660	0.9166	0.9402	1.952	3
	KNN	94.49	0.9935	0.8939	0.9405	1.90	8
	Decision tree	91.01	0.9109	0.9207	0.9065	3.103	3
	GaussianNB	91.44	0.9593	0.8595	0.9056	2.953	2

Bold values indicate the best model

5 The proposed federated learning model

All banks will first agree on a standard fraud detection global model (the model's architecture, activation function in each hidden layer, loss function, etc.). The existence of heterogeneity may lead to the misconvergence of the global model. Therefore, the proposed model requires handling the skewed data. The unbalanced data problem leads to the learned classifier identifying most of the fraud transactions as genuine ones. As a result, solving the unbalanced data issue is now a necessary step before developing a global model for fraud detection. Thus, the federated learning performance that is affected by statistical heterogeneity in the real-world scenario has been improved.

The proposed global model learns the fraud detection algorithm with the training data that is provided on its local database. Firstly, it handles the skewed data and normalizes the features in the appropriate interval. It then runs a neural network classification technique with an optimizer to obtain the optimal learning model parameters (gradients). Finally, it sends the gradients to the server. The combined global model has detected more fraud than each bank independently, even when the dataset is not shared. Figure 4 illustrates the main steps for the client–server process.

6 Experimental results

In this section, the impact of individual and hybrid resampling strategies on the dataset of credit card fraud detection is compared. Different classification methods, including DT; GaussianNB; RF; KNN, and Logistic Regression. The federated learning model has been built over multiple frameworks to preserve data security and privacy challenges.

The experiments in this work have been done using Python programming language (Python 3). In this work, we utilized open-source tools Scikit learn (1.1.3), pandas (1.4.4), NumPy (1.22.3), matplotlib (3.5.3), TensorFlow federated (0.17.0), PyTorch (1.2.0), and Imblearn (0.9.1) in this work. The experiment was carried out using a desktop computer with an Intel core i7 1.80 GHz CPU, 16GB of RAM, and Windows 10 64-bit operating system.

6.1 Dataset

The Kaggle dataset [43] used in this research contains actual but anonymous credit card transactions performed by European cardholders. The dataset includes 284,807 transactions made by credit card in September 2013 days. There is no missing data, and only 492 of the 284,807 transactions are fraudulent, resulting in a heavily skewed

Table 3 Comparison results of common machine learning classifiers after applying resampling techniques on credit card dataset

Resampling Method	Model	Accuracy	Precision	Recall	F1 score	Loss	Computation time (second)
ROS + RUS	Random forest	**99.98**	0.9995	1.0	**0.9997**	**0.0052**	**166**
	Logistic regression	98.60	0.9822	0.8621	0.9182	0.4815	12
	KNN	99.81	0.9945	1.0	0.9972	0.0624	132
	Decision tree	99.87	0.9961	1.0	0.9980	0.0439	12
	GaussianNB	93.39	0.9444	0.8519	0.8958	2.281	3
RUS + ROS	Random forest	99.89	0.9977	1.0	0.9988	0.0368	15
	Logistic regression	95.05	0.9739	0.9201	0.9461	1.707	3
	KNN	98.65	0.9742	0.9980	0.9860	0.4636	19
	Decision tree	99.06	0.9805	1.0	0.9901	0.3214	3
	GaussianNB	92.12	0.9656	0.8638	0.9117	2.718	2
Smote + ROS = = ROS + Smote	Random forest	**99.98**	0.9997	1.0	**0.9998**	**0.0036**	**2083**
	Logistic regression	94.51	0.9730	0.9156	0.9434	1.895	23
	KNN	99.91	0.9983	1.0	0.9991	0.0289	768
	Decision tree	99.82	0.9975	0.9989	0.9982	0.0597	247
	GaussianNB	91.43	0.9723	0.8530	0.9087	2.956	6
RUS + Smote	Random forest	98.89	0.9949	0.9830	0.9889	0.3804	43
	Logistic regression	94.66	0.9732	0.9186	0.9451	1.842	6
	KNN	98.39	0.9707	0.9979	0.9841	0.5556	42
	Decision tree	97.20	0.9640	0.9807	0.9722	0.9660	11
	GaussianNB	91.28	0.9725	0.8496	0.9086	3.008	8
Smote + RUS	Random forest	**99.98**	0.9997	0.9999	0.9998	**0.0039**	**3007**
	Logistic regression	94.51	0.9730	0.9156	0.9434	1.895	27
	KNN	99.91	0.9982	1.0	0.9991	0.0296	754
	Decision tree	99.82	0.9974	0.9990	0.9982	0.0603	246
	GaussianNB	91.44	0.9723	0.8530	0.9088	2.956	6
AdaSyn + ROS = = = ROS + AdaSyn	Random forest	**99.99**	0.9998	1.0	0.9999	**0.0017**	**1027**
	Logistic regression	94.61	0.9747	0.9159	0.9444	1.867	25
	KNN	99.96	0.9993	1.0	0.9996	0.0111	756
	Decision tree	**99.98**	**0.9996**	**1.0**	**0.9998**	**0.0059**	**75**
	GaussianNB	91.47	0.9708	0.8552	0.9093	2.943	6
RUS + AdaSyn	Random forest	93.18	0.9471	0.9153	0.9301	2.352	5
	Logistic regression	94.05	0.9642	0.9134	0.9372	2.052	3
	KNN	94.20	0.9813	0.8995	0.9379	2.002	7
	Decision tree	90.00	0.8854	0.9180	0.8995	3.453	3
	GaussianNB	90.14	0.9594	0.8306	0.8899	3.403	2
AdaSyn + RUS	Random forest	**99.98**	0.9997	1.0	0.9998	**0.0004**	**2504**
	Logistic regression	89.78	0.9077	0.8858	0.8966	3.527	42
	KNN	99.91	0.9982	1.0	0.9991	0.0301	785
	Decision tree	99.87	0.9980	0.9993	0.9987	0.0437	308
	GaussianNB	73.32	0.9268	0.5064	0.6549	9.214	8

Bold values indicate the best model

Table 4 Comparison between previous work [3, 44–47] and our work

Reference	Training and testing	NB	SVM	KNN	RF	DT	LR
[3] (2011)	–	–	93.8	–	96.2	–	94.7
[44] (2012)	–	96.04	–	–	91.09	–	–
[45] (2016)	–	94.10	94.17	–	95.81	95.19	–
[46] (2017)	(66: 34)	97.69	–	97.92	–	–	–
[46] (2017)	(90: 10)	97.52	–	97.15	–	–	–
[47] (2018)	(90: 10)	97.56	97.19	98.56	98.57	–	–
[47] (2018)	(66: 34)	97.70	97.39	97.97	98.25	–	–
[47] (2018)	(75: 25)	97.46	95.04	97.55	97.7	–	–
[47] (2018)	(80: 20)	97.80	97.46	98.16	98.23	–	–
AdaSyn + ROS (our work)	(80: 20)	91.47	–	99.96	99.99	99.98	94.61

Table 5 Comparison between Ata‘s work [47] and our work

Resampling technique	Performance measures of (80:20) data distribution			
	Classifier	Accuracy	Precision	Recall
Under-sampling [7]	NB	90.97%	96.55%	85.53%
	SVM	92.51%	94.18%	91.01%
	KNN	92.89%	97.00%	88.78%
	RF	93.02%	97.87%	89.03%
AdaSyn + ROS = = = ROS + AdaSyn	NB	91.47%	97.08%	85.52%
	RF	99.99%	99.98%	100%
	LR	94.61%	97.47%	91.59%
	KNN	99.96%	99.93%	100%
	DT	99.98%	99.96%	100%

dataset. Furthermore, it has 30 features, only two knowns, namely the transaction amount and time. See Table 1.

6.2 Results and discussion

In this section, performance metrics, including precision, recall, accuracy, loss, F1-measure, and total computational time have been discussed to ensure the effectiveness of all used classifiers in conjunction with Resampling techniques. For accuracy evaluation, the machine learning classification techniques and CNN classifier have been adopted for comparison. The comparative results have been done by using (80:20) training–testing ratio displayed that the.

6.2.1 Machine learning classifier with data balancing techniques

This section shows the experimental results of the individual and hybrid resampling techniques in conjunction with the common machine learning classifiers.

Table 2 shows that Smote with RF has the best result according to Accuracy, F1-score, and Loss but is the worst according to Time Computation. Therefore, ROS with DT is the best according to time computation and almost performance parameters. Then, hybrid resampling techniques were proposed to obtain more effective results, as shown in Table 3. This table shows that Oversampling, followed by Oversampling and Oversampling, followed by Undersampling in combination with RF classifier, is the best resampling strategy for class imbalance issues in all hybrid resampling methods.

Regarding each classifier's precision for various resampling methods, see Table 3. ROS + RUS; ROS + Smote, and ROS + AdaSyn routinely outperform the rest of these methods. Among machine learning classifiers, RF and DT have attained higher precision values.

The tribble equal operation that used in Table 3 means that the results of applying Smote then ROS as the same ROS then Smote.

For more reliability, the proposed hybrid approach is compared with many of the previous works, as shown in Tables 4 and 5. Wherever the proposed hybrid resampling technique is better than the previous works according to the performance measures.

The tribble equal operation that used in Table 5 means that the results of applying AdaSyn then ROS as the same ROS then AdaSyn.

In Table 4, the (80:20) distribution showed better performance for all common classifiers within the same data distribution. Table 5 shows that the proposed hybrid

Table 6 Cross-validation mean values for our study

Classifier/ Resampling techniques	SMOTE	AdaSyn	ROS	RUS	ROS + RUS	RUS + ROS	Smote + ROS	RUS + Smote	Smote + RUS	AdaSyn + ROS	RUS + AdaSyn	AdaSyn + RUS
Decision tree	0.9982	0.9986	0.9998	0.9101	0.9987	0.9906	0.9982	0.9720	0.9982	0.9998	0.900	0.9987
GaussianNB	0.9143	0.7332	0.9148	0.9144	0.9339	0.9212	0.9143	0.9128	0.9144	0.9147	0.9014	0.7332
KNeighbors	0.9991	0.9991	0.9996	**0.9449**	0.9981	0.9865	0.9991	0.9839	0.9991	0.9996	**0.9420**	0.9991
Random forest	**0.9998**	**0.9998**	**0.9999**	0.9391	**0.9998**	**0.9989**	**0.9998**	**0.9889**	**0.9998**	**0.9999**	0.9318	**0.9998**
Logistic regression	0.9451	0.8978	0.9457	0.9434	0.9860	0.9505	0.9451	0.9466	0.9451	0.9461	0.9405	0.8978

Bold values indicate the best model

oversampling technique (AdaSyn + ROS) is better than the individual undersampling techniques for different classifiers.

Cross-validation is a very useful statistical approach for evaluating machine learning models several times to detect overfitting. In this study, k-fold cross-validation has been used. Grid search cross-validation method selected k = 10, on the given scale, our study will predict the highest accuracy, especially for RF with most resampling techniques. The cross-validation mean values of our study are compared with the previous work [48], which presents the Cross-Validation mean values obtained by each classifier for different resampling techniques. The outcomes of Tomeklinks (TMLK), ROS, and SMOTE techniques are as follows (0.964, 0.970, and 0.973). In our study. Among all resampling techniques, ROS and ROS + AdaSyn techniques have achieved excellent results. Among classifiers, RF has achieved higher mean values (0.9999). Table 6 presents the Cross-Validation mean values acquired by each classifier for various resampling approaches.

Each classifier's overall evaluation time includes both the training and testing phases. Table 7 shows the total time of our study. This table shows that the total time of the previous work [48] is better than our study because this work used PCA as a preprocessing step. Due to a large amount of data, RF takes longer when combined with oversampling techniques. Although undersampling techniques save time, they may result in underfitting.

Finally, the proposed hybrid resampling techniques within machine learning classifiers outperformed the individual resampling technique.

6.2.2 CNN classifier with data balancing techniques

The following is the structure of a CNN used to detect credit card fraud:

A vector of attributes that characterize each transaction is taken by the input layer. After applying 28 filters in the first convolutional layer, an activation function known as a rectified linear unit (ReLU) is applied. From the data, this layer extracts low-level information like transaction frequency or spending trends. 32fliter is applied by the second convolutional layer, and then there is another ReLU activation function.

After applying a 64-filter in the third convolutional layer, there is another ReLU activation function. From the data, this layer extracts higher-level traits like anomalies and outliers. The max pooling layer reduces the dimensionality of the second layer's output. This layer aids in the reduction of overfitting and the enhancement of generalization. The flatten layer transforms the max pooling layer's output into a one-dimensional vector that can be fed into a fully connected layer.

The output of the flatten layer is applied to the fully connected layer, which then performs a linear transformation followed by a dropout operation. The dropout operation, with a probability of 0.5, randomly sets some of the units to zero, which also helps to prevent overfitting and improve robustness. The output layer takes the fully connected layer's output and applies the sigmoid function to generate a probability distribution over two classes: fraud or not.

This section shows the experimental results of the individual and hybrid resampling techniques in conjunction with the CNN classifiers. The hybrid resampling techniques performed very well on machine learning classifiers. On another hand, the individual resampling techniques performed better with CNN classifier than the hybrid resampling techniques, as shown in Table 8. Among all resampling techniques, ROS and Smote have achieved higher accuracy values (99.93%).

For a fair comparison, the same hyper-parameters for previous works are used. There is no existing work that

Table 7 Total time values for imbalanced methods of our study

Classifier/ resampling techniques	Smote	AdaSyn	ROS	RUS	ROS + RUS	RUS + ROS	Smote + ROS	RUS + Smote	Smote + RUS	AdaSyn + ROS	RUS + AdaSyn	AdaSyn + RUS
Decision tree	270	345	84	3	12	3	247	11	246	75	3	308
GaussianNB	**6**	**10**	**8**	**2**	**3**	**2**	**6**	8	**6**	**6**	**2**	**8**
KNeighbors	814	821	754	8	132	19	768	42	754	756	7	785
Random forest	2240	2683	1090	6	166	15	2083	43	3007	1027	5	2504
Logistic regression	34	40	32	3	12	3	23	**6**	27	25	3	42

Bold values indicate the best model

provides the same level of efficiency. For class imbalance problems, the smote resampling strategy works best with CNN. To ensure this result, the Smote + CNN is compared to two prior studies [48, 49], the results of which are presented in Tables 9 and 10.

Table 9 compares the proposed Model (Smote + CNN) to the baseline state-of-the-art models [49]. We can see that the proposed Model (Smote + CNN) outperforms the state-of-the-art ensemble. In terms of the majority of the criteria, particularly the F1 measure, LSTM, GRU, and ensemble model ℓ are used as ensemble techniques. The model reached its greatest AUC-ROC score, demonstrating the ability of the proposed model to distinguish between fraudulent and normal transactions as well as its ability to work with extremely unbalanced data.

The Smote + CNN algorithm outperformed all compared models, as shown in Table 11. In Table 12, the CNN model has been built with the same framework architecture of the previous work in addition to the resampling step using Smote resampling techniques. The simulation results showed that the proposed Smote + CNN model is better than the traditional CNN model. CNN classifier with data balancing techniques.

6.2.3 Federated learning model with different batch sizes over several frameworks

A federated learning model has been built with different batch sizes to select the optimal number of batch sizes for the CCFD problem, as shown in Table 11; then, this model runs on different environments, and these results are presented in Tables 12 and 13.

As per the graphical representation of these boxplots, shown in Figs. 5, 6, 7, 8, 9, for different classification techniques (machine learning classifier and CNN classifier) in combination with different resampling (individual and hybrid) techniques. For each resampling technique, the performance of all used classifiers is presented.

Tables 12 and 13 present the performance of the traditional model of Smote + CNN on Tensorflow and PyTorch environments, respectively. The accuracy of the traditional model on TensorFlow is better than PyTorch for all optimizers. However, it costs more computational time. On the side, the performance of the federated model of Smote + CNN on Tensorflow federated and PyTorch-pysyft environments, respectively. The accuracy of the federated model on PyTorch-pysyft is better than TensorFlow federated for most optimizers. But it requires more computational time.

The federated learning results are applied using 100 iterations, and Adam optimizer has used learning rate 0.1, SGD 0.1, and MSGD 0.1 with 0.2 as moment value.

Table 8 Comparison results of CNN classifier

Resampling method	Model	Accuracy	Precision	Recall	F1 score	Loss	Computation time (s
ROS	CNN	99.93	0.8082	0.8027	0.8054	0.0230	259
Smote		99.93	0.8263	0.8095	0.8178	0.0214	255
AdaSyn		99.91	0.7283	0.8027	0.7637	0.0295	250
RUS		98.33	0.0843	0.8775	0.1538	0.5736	17
ROS + RUS		99.84	0.5294	0.8571	0.6545	0.0537	52
RUS + ROS		99.92	0.7439	0.8299	0.7845	0.0270	145
Smote + ROS = = ROS + Smote		99.92	0.7579	0.8095	0.7828	0.0266	257
RUS + Smote		98.77	0.1127	0.8911	0.2001	0.4232	20
Smote + RUS		99.90	0.7034	0.8231	0.7586	0.0311	253
AdaSyn + ROS = = = ROS + AdaSyn		99.92	0.7393	0.8299	0.7820	0.0274	251
RUS + AdaSyn		98.35	0.0846	0.8707	0.1543	0.5671	16
AdaSyn + RUS		99.92	0.7530	0.8299	0.7896	0.0262	258

Table 9 Comparison between previous work [49] and our work

Model	Precision	Recall	F1 Score	AUC-ROC
GRU	0.8626	0.7208	0.7792	0.8602
LSTM	0.8575	0.7408	0.7866	0.8702
Ensemble model ℓ	**0.9569**	0.6674	0.7813	0.8337
Smote + CNN	0.8263	**0.8095**	**0.8178**	**0.9378**

Bold values indicate the best model

Figures 10, 11, 12 demonstrate the performance of the single CNN model with different optimization techniques whereas Fig. 10 displays how the Adam optimizer outperforms other optimizers, especially on tensorflow platform. Tensorflow platform is faster than Pytorch platform with all optimization techniques as shown in Fig. 13. In Fig. 12, the Adam optimizer achieves the minimum loss value on Pytorch platform.

Figures 13, 14, 15 demonstrate the performance of the federated model with different optimization techniques whereas Fig. 13 displays how MSGD optimizer outperforms other optimizers. TensorFlow federated platform is faster than Pytorch_pysyft platform with all optimization techniques as shown in Fig. 14. In Fig. 15, the MSGD optimizer achieves the minimum loss value on Tensorflow Federated platform.

Table 10 Comparison between previous work [48] and our work

Model	Framework-architecture	Hyper-parameters	Accuracy	Categorical prediction accuracy (%)		False Positive	FCR (%)
				Legitimate	Fraudulent		
CNN [48]	Output-layer	2Classes; Softmax	99.89	**99.82**	82	17	82
	Loss function	MAE					
	Optimizer	SGD					
	Epochs	10					
	Splitting ratio	80:20					
Smote + CNN	Output-layer	2Classes; Softmax	**99.93**	99.78	**83**	17	**84**
	Loss function	MAE					
	Optimizer	SGD					
	Epochs	10					
	Splitting ratio	80:20					

Bold values indicate the best model

Table 11 Results of different batch sizes of FL_SMOTE + CNN

FL_Smote + CNN						
Optimizer	ADAM					
	Accuracy	Precision	Recall	F1 Score	Loss	Time
Batch size = 32	92.14	0.8799	0.9759	0.9254	**0.2191**	421
Batch size = 64	**92.15**	0.8708	**0.9865**	**0.9255**	0.2215	291
Batch size = 128	89.21	0.8341	0.9789	0.9007	0.3344	266
Batch size = 256	80.91	0.7271	0.9855	0.8368	0.5602	**206**

Bold values indicate the best model

Table 12 Comparison results between the traditional model and federated model over TensorFlow environment

Traditional_Smote + CNN									
Framework	Tensorflow								
Optimizer	ADAM			SGD			MSGD		
	Accuracy	Loss	Time	Accuracy	Loss	Time	Accuracy	Loss	Time
Single Model	**99.92**	**0.0266**	**266**	99.72	0.0933	295	99.46	0.0334	266
Federated_Smote + CNN									
Framework	Tensorflow Federated								
Optimizer	ADAM			SGD			MSGD		
	Accuracy	Loss	Time	Accuracy	Loss	Time	Accuracy	Loss	Time
Federated Model	92.15	0.2215	291	91.97	0.3098	355	**92.93**	**0.2120**	**284**

Bold values indicate the best model

Table 13 Comparison results between the traditional model and federated model over the PyTorch environment

Traditional_Smote + CNN									
Framework	Pytorch								
Optimizer	ADAM			SGD			MSGD		
	Accuracy	Loss	Time	Accuracy	Loss	Time	Accuracy	Loss	Time
Single Model	**99**	**0.0090**	**15**	97	0.1099	12	97	0.1072	14
Federated_Smote + CNN									
Framework	Pysyft								
Optimizer	ADAM			SGD			MSGD		
	Accuracy	Loss	Time	Accuracy	Loss	Time	Accuracy	Loss	Time
Federated Model	**93**	0.2658	488	92	**0.2245**	**401**	90	0.3229	444

Bold values indicate the best model

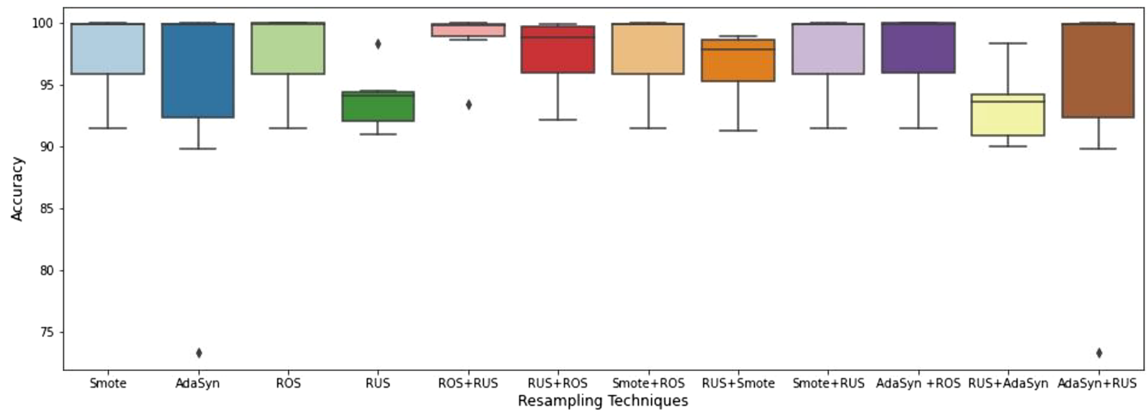


Fig. 5 BoxPlot of accuracy

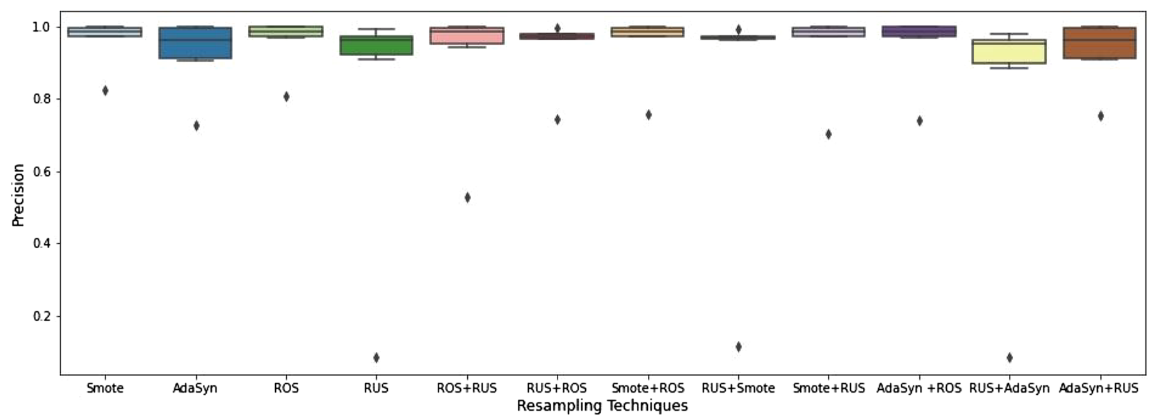


Fig. 6 BoxPlot of precision

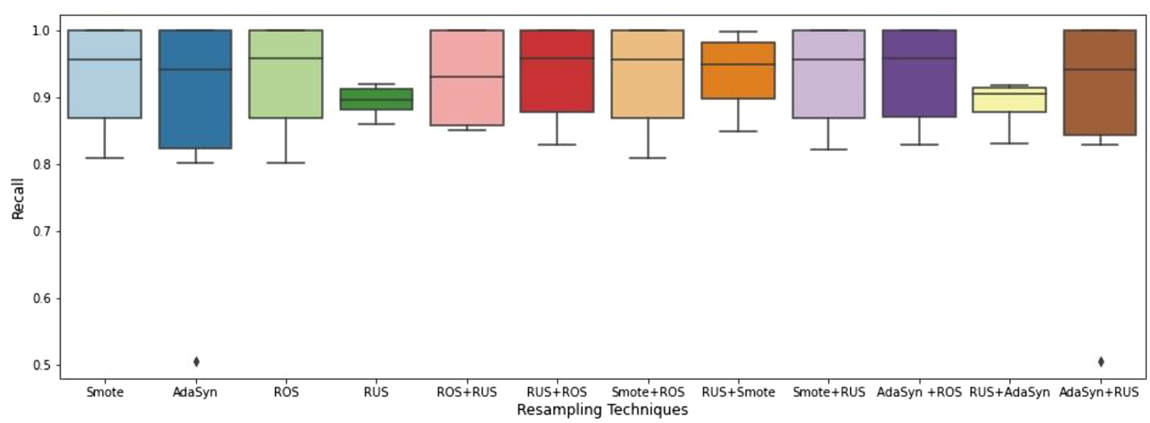


Fig. 7 BoxPlot of recall

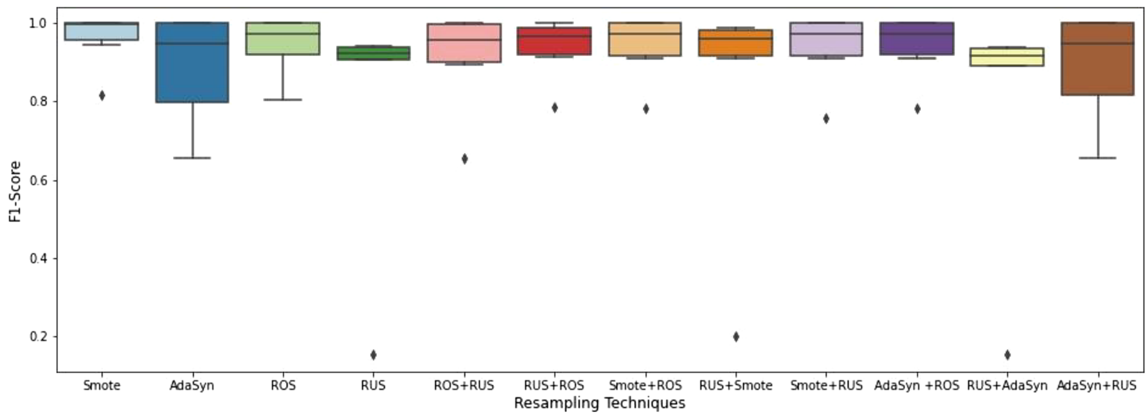


Fig. 8 BoxPlot of f1-score

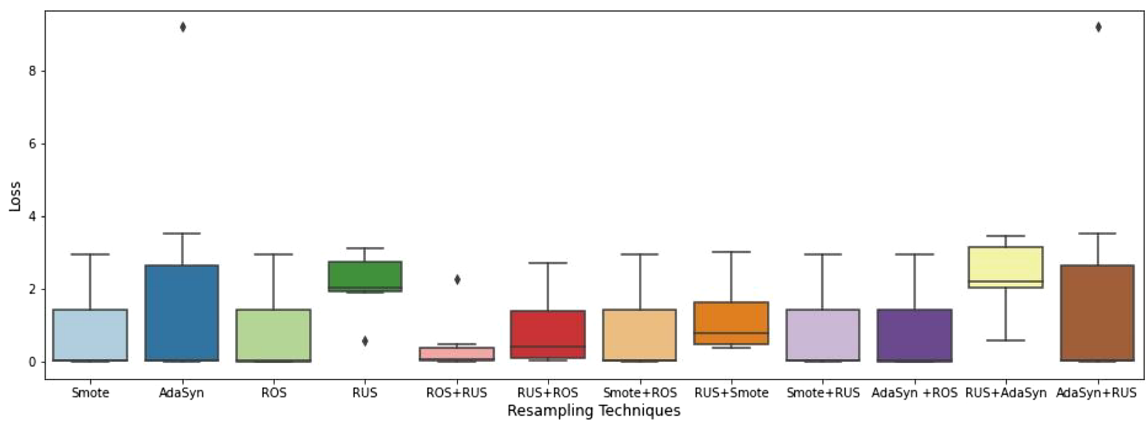


Fig. 9 BoxPlot of loss

Fig. 10 Accuracy of the single model across tensorflow and pytorch platforms

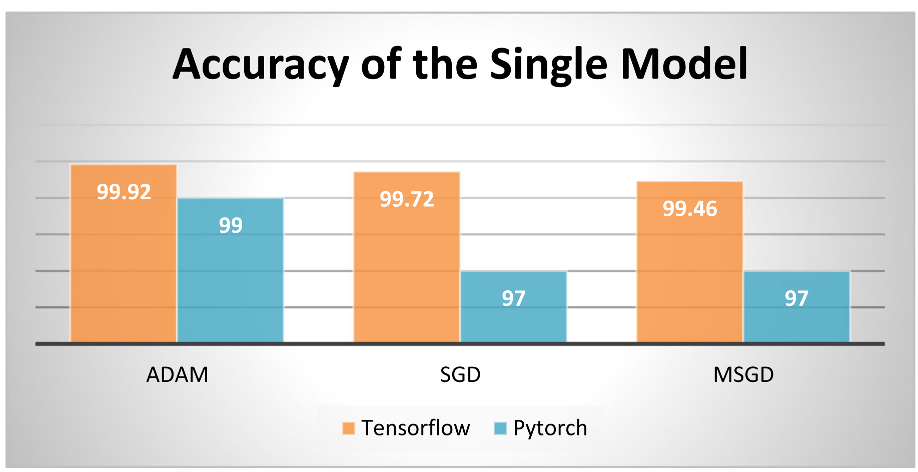


Fig. 11 Time of the single model across tensorflow and pytorch platforms

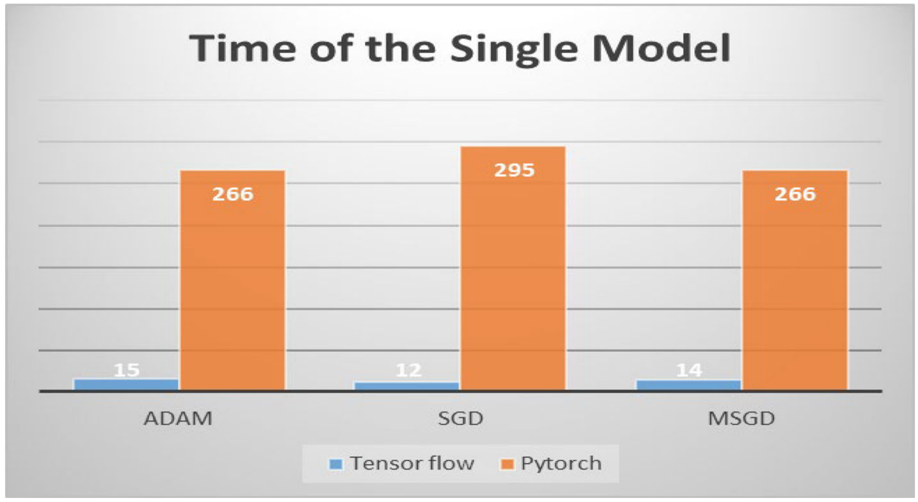


Fig. 12 Loss of the single model across tensorflow and pytorch platforms

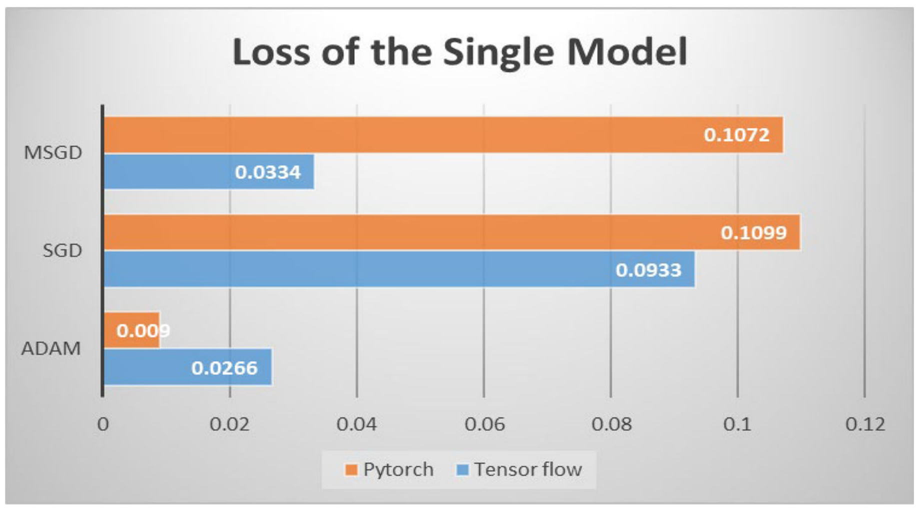
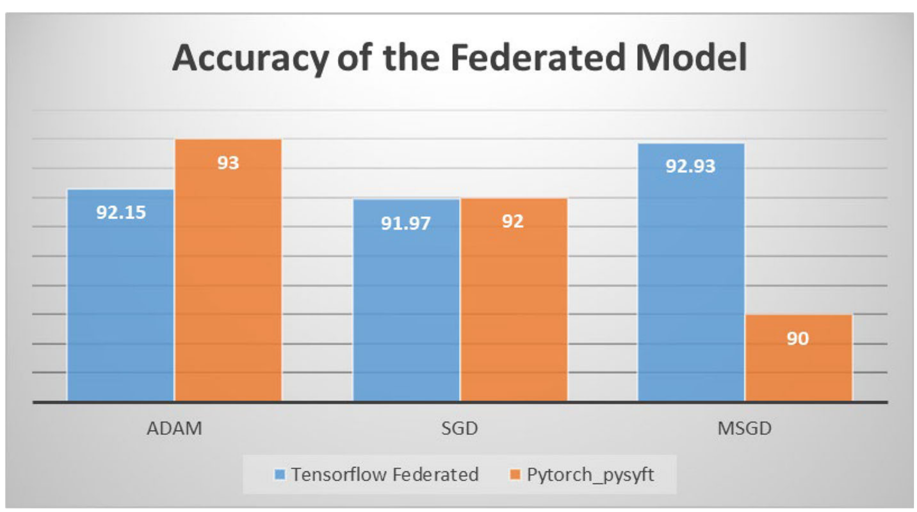


Fig. 13 Accuracy of the federated model with different optimizers



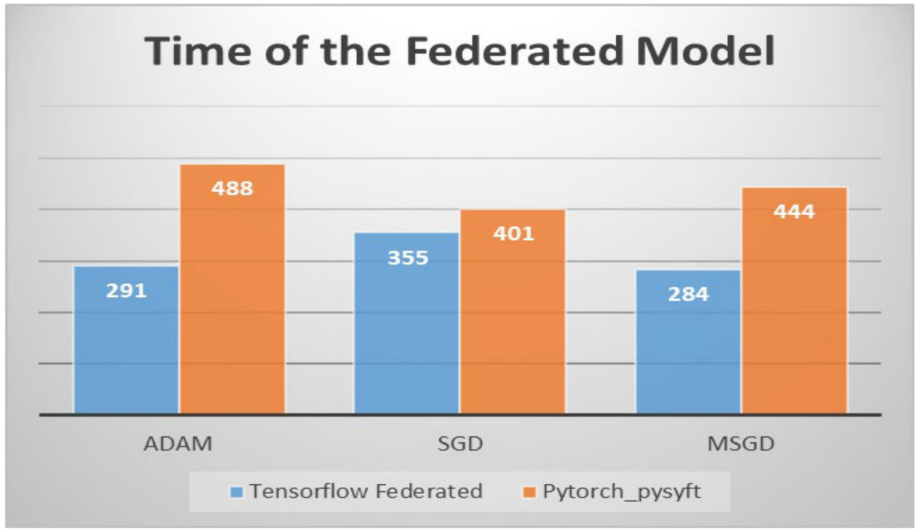


Fig. 14 Time of the federated model with different optimizers

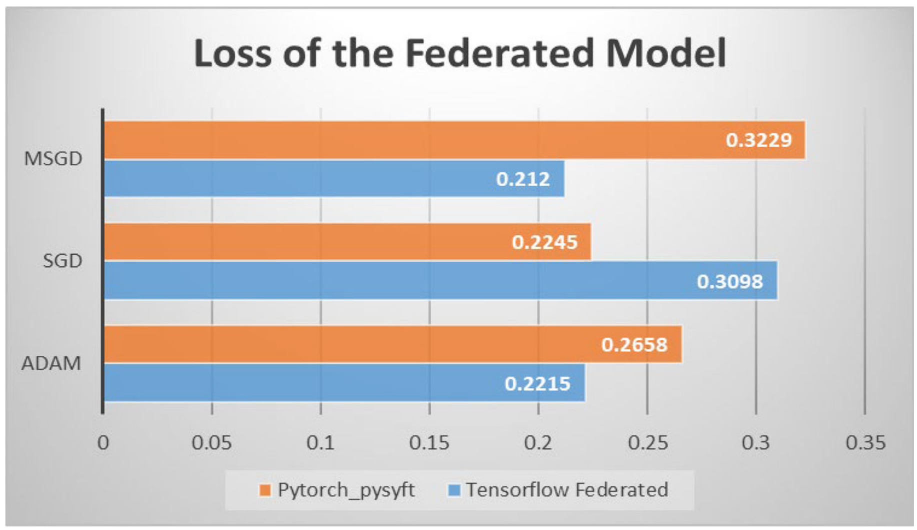


Fig. 15 Loss of the federated model with different optimizers

7 Conclusion

A federated learning approach for CCFD is presented in this research to address data privacy concerns. Additionally, hybrid resampling methods were suggested as a way to address imbalanced class issues and enhance classification efficacy. The outcomes of the experiments demonstrated that when combined with the proposed federated learning approach. Notably, the Smote resampling technique is the best with the proposed CNN model, and AdaSyn + ROS is the best with the DT model according to all performance parameters and computational time. The accuracy of the federated model on PyTorch-pysyft (93%, 92%, 90%) is better than TensorFlow federated (92.15%, 91.97%, 92.93%) for Adam, SGD and MSGD optimizers, respectively. However, it costs more computational time. Because of the dataset's limitations, this should be approached with caution. The best accuracy for the RF, LR, KNN, DT, and Gaussian NB classifiers is 99,99%; 94,61%; 99.96%; 99,98%; and 91,47%, respectively, according to the experimental data. The comparative results reveal that the RF outperforms the NB, RF, DT, and KNN with high performance characteristics (accuracy, recall, precision, and f score). With all resampling approaches, RF achieves the lowest loss levels.

In future works, the performance of the proposed federated learning model will be improved by integrating more advanced optimization techniques. Also, privacy protection of gradients (learning parameters) that may lead to model poisoning by injecting malicious data will be handled. The federated model's communication and aggregation updates will be optimized in a secure and scalable way.

Funding Open access funding provided by The Science, Technology & Innovation Funding Authority (STDF) in cooperation with The Egyptian Knowledge Bank (EKB). The authors received no specific funding for this study.

Data availability Data are available from the authors upon reasonable request.

Declarations

Conflict of interest The authors declare that they have no conflicts of interest to report regarding the present study.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

Informed consent Informed consent was obtained from all individual participants included in the study.

References

1. NilsonReport.Card Fraud Losses Reach $27.85 Billion (2019) https://nilsonreport.com/mention/407/1link/ Accessed 16 Jun 2021
2. Makki S et al (2019) An experimental study with imbalanced classification approaches for credit card fraud detection. IEEE Access 7:93010–93022
3. Awoyemi JO, Adetunmbi AO, Oluwadare SA (2017) Credit card fraud detection using machine learning techniques: a comparative analysis. In: 2017 international conference on computing networking and informatics (ICCNI). IEEE. Johar Town, Lahore, Punjab 54770, Pakistan pp 1–9
4. Dornadula VN, Geetha S (2019) Credit card fraud detection using machine learning algorithms. Procedia Comput Sci 165:631–641
5. Naik H, Kanikar P (2019) Credit card fraud detection based on machine learning algorithms. Int J Comput Appl 182(44):8–12
6. Khare N, Sait SY (2018) Credit card fraud detection using machine learning models and collating machine learning models. Int J Pure Appl Math 118(20):825–838
7. Banal A, Garg H (2021) An efficient techniques for fraudulent detection in credit card dataset: a comprehensive study. In: IOP conference series: materials science and engineering. Mathura, India, 1116(1). IOP Publishing
8. Zhang W, Weishan T et al (2021) Dynamic fusion-based federated learning for COVID-19 detection. IEEE Internet Things J 8(21):15884–15891
9. Lian X et al. (2017) Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. Adv Neural Inf Process Syst 30
10. Abd Elrahman SM, Abraham A (2013) A review of class imbalance problem. J Netw Innov Comput 1(2013):332–340
11. Bejjanki G, Jayadev G, Narsimha G (2018) Class processing and systems. Springer
12. Liu Y, Li X, Chen X, Wang X, Li H (2020) High-performance machine learning for large-scale data classification considering class imbalance. Sci Program
13. Zheng W, Jin M (2020) The effects of class imbalance and training data size on classifier learning: an empirical study. SN Comput Sci 1(2):1–13
14. Sweers T, Heskes T, Krijthe J (2018) Autoencoding credit card fraud. Bachelor Thesis
15. Xuan S et al. (2018) Random forest for credit card fraud detection. In: 2018 IEEE 15th international conference on networking, sensing, and control (ICNSC). IEEE, China
16. Singh G et al (2012) A machine learning approach for detection of fraud based on svm. Int J Sci Eng Technol 1(3):192–196
17. Sonawane YB, Gadgil AS, More AE, Jathar NK (2016) Credit card fraud detection using clustering based approach. Int J Adv Res Innov Ideas Educ 2(6)
18. Xie X et al. (2018) Generative adversarial network-based credit card fraud detection. In: International conference in communications, signal processing and systems. Springer, Singapore
19. Niu X, Wang L, Yang X (2019) A comparison study of credit card fraud detection: supervised versus unsupervised. arXiv preprint arXiv:1904.10604
20. Fahmi M, Hamdy A, Nagati K (2016) Data mining techniques for credit card fraud detection: empirical study. Sustain Vital Technol Eng Inf, pp 1–9
21. Chen K, Seshadri S, Zhang LJ (2019) Big Data–BigData 2019: 8th international congress, Held as part of the services conference federation, SCF 2019, San Diego, CA, USA, June 25–30, Proceedings. Vol. 11514. Springer
22. Y. Wensi et al. (2019) Ffd: a federated learning based method for credit card fraud detection. J Big Data, LNCS 11514, pp 18–32
23. Suvarna R, Meena Kowshalya A (2020) Credit card fraud detection using federated learning techniques. J Web Eng Technol 7(3):356–367
24. Albertio C (2019) Towards Efficient and Privacy-preserving Federated Deep Learning. In: International conference on science and technology on communication security laboratory, 978-I-5386–8088- 9/19@IEEE
25. Lim WYB et al (2020) Federated learning in mobile edge networks: a comprehensive survey. IEEE Commun Surv Tutor 22(3):2031–2063
26. Yao X, Huang T, Wu C, Zhang R, Sun L (2019) Towards faster and better federated learning: a feature fusion approach. In: 2019 IEEE international conference on image processing (ICIP). IEEE, Taipei, Taiwan, pp175–195
27. Panigrahi S et al (2009) Credit card fraud detection: a fusion approach using Dempster-Shafer theory and Bayesian learning. Inf Fusion 10(4):354–363
28. Khan MZ, Pathan JD, Ahmed AHE (2014) Credit card fraud detection system using hidden markov Model and K-clustering. Int J Adv Res Comput Commun Eng 3(2):5458
29. Kundu A, Panigrahi S, Sural S, Majumdar AK (2009) Blast-ssaha hybridization for credit card fraud detection. IEEE Trans Dependable Secure Comput 6(4):309–315
30. Galar M, Fernandez A, Barrenechea E, Bustince H, Herrera F (2011) A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. IEEE Trans Syst Man Cybern Part C Appl Rev 42(4):463–484
31. Japkowicz N, Shah M (2011) Evaluating learning algorithms: a classification perspective. Cambridge University Press

32. Huang C, Li Y, Loy CC, Tang X (2019) Deep imbalanced learning for face recognition and attribute prediction. EEE Trans Pattern Anal Mach Intell 42(11):2781–2794
33. Ouyang X, Chen Y, Wei B (2017) Experimental study on unbalanced data problem using an oil spill training data set. J Adv Math Comput Sci 21:1–9
34. Yang P et al (2013) Sample subset optimization techniques for imbalanced and ensemble learning problems in bioinformatics applications. IEEE Trans. Cybern. 44(3):445–455
35. Sun B, Chen H, Wang J, Xie H (2018) Evolutionary under-sampling-based bagging ensemble method for imbalanced data classification. Front Comput Sci 12(2):331–350
36. Kamaruddin S, Ravi V (2016) Credit card fraud detection using big data analytics: use of PSOAANN based one-class classification. In: Proceedings of the international conference on informatics and analytics, Pondicherry India, pp 1–8
37. Wei W et al (2013) Effective detection of sophisticated online banking fraud on extremely imbalanced data. World Wide Web 16(4):449–475
38. N.D. Stout. Undersampling and Oversampling Statistics Visual Example. Pinterest. https://www.pinterest.it/pin/514958538641697615/
39. Ling CX, Li C (1998) Chenghui. Data mining for direct marketing: Problems and solutions. In: Kdd, pp 73–79
40. Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP (2002) SMOTE: synthetic minority over-sampling technique. J Artif Intell Res 16:321–357
41. He et al H (2008) AdaSyn: adaptive synthetic sampling approach for imbalanced learning. In: 2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence). IEEE, Hong Kong, pp.1322–1328
42. Fernández A et al (2018) SMOTE for learning from imbalanced data: progress and challenges, marking the 15th anniversary. J Artif Intell Res 61:863–905
43. Machine Learning Group—ULB (2018) Credit card fraud detection anonymized credit card transactions labeled as fraudulent or genuine. https://www.kaggle.com/mlg-ulb/creditcardfraud
44. Bhattacharyya S et al (2011) Data mining for credit card fraud: a comparative study. Decis Support Syst 50(3):602–613
45. Alowais MI, Soon LK (2012) Credit card fraud detection: Personalized or aggregated model. In: 2012 third FTRA international conference on mobile, ubiquitous, and intelligent computing. IEEE, Vancouver, Canada, pp 114–116
46. Kültür Y, Mehmet UC (2017) Hybrid approaches for detecting credit card fraud. Expert Syst 34(2):e12191
47. Ata O, Hazim L (2020) Comparative analysis of different ˘distributions dataset by using data mining techniques on credit card fraud detection. Tehnicki vjesnik 27(2):618–626
48. Singh A, Ranjan RK, Tiwari A (2021) Credit card fraud detection under extreme imbalanced data: a comparative study of data-level algorithms. J Exp Theor Artif Intell 34:1–28
49. Forough J, Momtazi S (2021) Ensemble of deep sequential models for credit card fraud detection. Appl Soft Comput 99(2):106883