Detection Optimization of Rare Attacks in Software-Defined Network using Ensemble Learning

Ahmed M. El-Shamy  
*Canadian International College, Cairo, Egypt*, ahshamy@gmail.com

Nawal A. El-Fishawy  
*Computer Science and Engineering Department at Faculty of Electronic Engineering, Menoufia University, Cairo, Egypt*

Gamal M. Attiya  
*Computer Science and Engineering Department at Faculty of Electronic Engineering, Menoufia University, Cairo, Egypt*

Mokhtar Ahmed  
*Computer Science and Engineering Department Faculty of Electronic Engineering, Menoufia University, Cairo, Egypt*

Follow this and additional works at: [https://mej.researchcommons.org/home](https://mej.researchcommons.org/home)

Part of the Digital Communications and Networking Commons, and the Other Computer Engineering Commons

**Recommended Citation**

Available at: [https://doi.org/10.58491/2735-4202.3020](https://doi.org/10.58491/2735-4202.3020)

This Original Study is brought to you for free and open access by Mansoura Engineering Journal. It has been accepted for inclusion in Mansoura Engineering Journal by an authorized editor of Mansoura Engineering Journal. For more information, please contact mej@mans.edu.eg.
Detection Optimization of Rare Attacks in Software-Defined Networking Using Ensemble Learning

Ahmed M. El-Shamy a,*, Nawal A. El-Fishawy b, Gamal M. Attiya b, Mokhtar A. A. Mohamed b

a Business Technology Department, Canadian International College CIC, Cairo, Egypt
b Department of Computer Science and Engineering, Faculty of Electronic Engineering, Menoufi University, Menoufi, Egypt

Abstract

Software-defined networking (SDN) is a highly flexible architecture that automates and facilitates network configuration and management. Intrusion detection systems (IDS) are becoming essential components in the network to detect malicious attacks and suspicious activities by continuously monitoring network traffic. Integration between SDN and machine learning (ML) techniques is extensively used to build an effective IDS against all potential cyberattacks that aim at breaking the network security policy and stealing valuable data. Implementing an IDS based on SDN and ML has the advantage of managing traffic dynamically and fully autonomously to provide high protection against security threats. The main objective of this paper is to propose an IDS based on SDN that integrates adaptive boosting and cost-sensitive techniques to improve the detection rates of rare attacks without compromising the detection accuracy of familiar attacks. Cost-matrix values for various attacks are optimized using grid search and genetic algorithms. The integration of AdaBoost and cost-sensitive aims to build a classification model that minimizes the total number of high-cost errors caused by incorrectly classifying attacks. Experiments were conducted on the CSE–CIC–IDS2017 and CSE–CIC–IDS2018 datasets to simulate the network. The performance of the proposed algorithm is evaluated using the macro F1-score and the geometric average of the recall (G-mean). Simulation results prove that the proposed algorithm enhances the detection performance of rare intrusions compared with current techniques, where it achieves a macro F1-score of 0.99 and a G-mean of 0.992 using CIC-IDSC-IDS2017, while for CIC-IDSC-IDS2018 the macro F1-score is 0.993 and the G-mean is 0.994.

Keywords: AdaBoost ensemble learning, Cost-sensitive learning, Imbalanced classes, Intrusion detection, Rare attacks, Software-defined networking

1. Introduction

Cloud data centers host many applications for the numerous advantages they provide to users, such as high availability, lower cost, and security. However, cloud data center networks are prone to cyberattacks by intruders, who attempt to break through and bypass security measures to illegally access data. In today’s world, security vulnerabilities and breaches are rising significantly due to technological advances and the growth of applications and services hosted in cloud-based data centers. All of these have led to a tremendous increase in security attacks (Chang et al., 2022).

Securing cloud data center networks against various security threats is becoming increasingly difficult and complex. A cloud data center develops a robust defense-in-depth security policy in which many layers of security technology are implemented within the network. These security measures include firewalls, antivirus, encryption techniques, access control lists, intrusion detection systems...
(IDS), and intrusion prevention systems, as depicted in Fig. 1. The main objectives of the defense-in-depth security approach are to ensure the availability, integrity, and confidentiality of the data and applications in data centers (Nguyen and Debroy, 2022).

A firewall is the primary security device on the cloud data center networks to protect valuable data against harmful attacks. However, security solutions using firewalls depend on the network administrator configuring the firewalls with static rules to allow or block network traffic. In addition, these rules must be updated to reflect any changes to the security policy (Williams et al., 2021). Intrusion detection and prevention systems (IDPS) are widely used in network security solutions due to their vital role in securing and protecting the cloud data center network infrastructure. IDPS continuously monitors the network traffic and identifies any malicious activities that may compromise network security. Whenever potential security vulnerability is discovered, it automatically prevents this suspicious traffic. In addition, it sends an alert to the administrator to indicate the existence of a security breach and stores it in a central security information management system for further analysis (Xiao et al., 2021).

Layers of security defense-in-depth model (Dang, 2019).

Existing IDS can efficiently identify familiar and well-known attacks that frequently occur in the cloud data center. However, it faces a big challenge as it cannot accurately detect rare attacks where it produces a high false alarm rate. Rare attacks are unfamiliar types of security attacks with a low occurrence rate that may penetrate the victim's network and cause dangerous effects such as stealing valuable data, consuming the available resources, or controlling the network (Wisanwanichthan and Thammawichai, 2021).

Integration between software-defined networks (SDN) and machine learning (ML) techniques is extensively used to build an effective IDPS against all potential cyberattacks in the cloud data center (Kranthi et al., 2022). SDN is an emerging technology that facilitates network management and offers attractive functionalities such as flexibility, programmability, and efficient resource management. Although SDN provides significant benefits to the cloud data center, it can be a target for potential security attacks. Building IDS based on SDN and ML has the advantage of managing traffic dynamically and fully autonomous to provide high protection with other security tools against threats and vulnerabilities (Deb and Roy, 2022).

The centralized architecture and network programmability of SDN may open a door for new attacks that did not exist in the traditional data center architecture. Attacks on the SDN have the potential to have devastating effects on the entire data center network. On the contrary, in the traditional network architecture, an attacker could only get unauthorized access to a single server or network subnet; in order to attack other servers or remaining network subnets, the attacker would need to escalate the acquired privilege. Therefore, it is essential to implement efficient and effective security mitigation techniques to defend against such harmful attacks promptly (Rahouti et al., 2022).

![Fig. 1. Layers of security defense-in-depth model (Dang, 2019).](image-url)
Zero-day attacks are rapidly emerging because of the attack techniques that become increasingly sophisticated every day.

The main problem with the ML algorithms used in security solutions is detecting rare attacks, which have few samples or undefined patterns in the collected security dataset. Rare attacks appear as minority classes, and the collected datasets become imbalanced where the distribution of dataset classes is not uniform. Detection of rare attacks seems challenging for classification algorithms due to the unbalanced class distribution in most security benchmark datasets. This problem limits the performance of a classifier for minority classes. In such critical situations, the used ML classifier is biased toward the majority class. Therefore, the classifier's performance will be better at correctly predicting majority class cases than minority classes, where rare attacks are either undetected as they may be classified as benign traffic or misclassified as another category of security threat. Undetected attacks pose a greater risk to the network's valuable resources and data (Macas et al., 2022). Therefore, several prior approaches have been implemented in the IDS design strategies to enhance the detection rate of rare attacks. Some of these solutions used data preprocessing methods such as resampling using undersampling or oversampling techniques to balance the datasets.

The main motivation for this paper is to overcome the drawbacks and limitations of the current studies, which are as follows.

The resampling-based methods have the potential to introduce a bias into the classification results because they modify the distributions and characteristics of the data (Ahsan et al., 2021). Some IDS-based approaches assume that the distribution of the dataset classes is balanced. As a result, their classification performance suffers, and they give suboptimal results when deployed with unbalanced datasets. They produce high detection accuracy for a specific category of attacks while performing poorly for other types of cyber threats. In addition, their assumption does not apply to most security datasets (Yu et al., 2022).

Other solutions only try to improve the overall detection rate while ignoring the impact of IDS performance on the detection rate of rare attacks. Finally, most existing IDS algorithms evaluate the effectiveness of rare attack detection by calculating accuracy based on the percentage of correctly classified instances, which is misleading. For example, if the class distribution ratio for a dataset consisting of two categories is 98:2 and the classification accuracy of the used ML algorithm is 98%, by simply detecting all instances as majority class. Then this classification accuracy percentage is accepted as the effect of minority classes on overall accuracy is very small. But this performance is deceptive because the minority samples are totally missing, and the ML algorithm failed to identify these attacks (Liao et al., 2022).

The big challenge to security solutions in the operational network is that the attack packets have a high level of stealth within the benign traffic and have fewer packets than normal traffic. Attacks are classified based on the rate of occurrence into familiar attacks and rare attacks. The rare attacks appear in the collected dataset as a minority class as they have a small number of instances compared with the other categories, which have a large number of samples and are called the majority classes; this causes the dataset to be imbalanced (Ran et al., 2022). The rare attacks are risky because most current IDS fail to capture such attack patterns with adequate reliability (Staddon et al., 2021).

The main contribution of this paper is to propose an SDN-based algorithm that integrates adaptive boosting (AdaBoost) and cost-sensitive methods to improve the detection rates of rare attacks without compromising the detection accuracy of familiar attacks. The cost matrix for various attack types is determined using two different optimization algorithms: grid search and genetic algorithms. AdaBoost and cost-sensitive methods aim to build a classification model that generates the lowest misclassification cost by reducing the total number of high-cost errors from the incorrect classification of rare attacks. The experiments were conducted on the CSE−CIC−IDS2017 and CSE−CIC−IDS2018 datasets to simulate the cloud data center. The proposed approach's performance is analyzed using the macro F1-score and the geometric average of recall (G-mean).

The rest of this paper is organized as follows. Section II presents an overview of the security threats in SDN architecture, IDS types, and ML schemes. Section III reviews in detail some of the literature reviews on the topic of interest. Section IV provides the architecture of the proposed IDS solution. Section V discusses the experimental results. Finally, conclusions are drawn, and directions for future work are suggested in Section VI.

2. Background

This section presents a concise overview of the existing threat vectors in the SDN environment,
discusses the different types of IDS, and finally explains the ensemble learning techniques used to build the proposed model.

2.1. The existing threat vectors in software-defined networking

SDN is a highly flexible network technique that enables automated network provisioning and policy-based network management. SDN technology separates the control plane that makes decisions from the forwarding plane within the network switches and routers. A centralized SDN controller monitors and manages the entire network (Shagghahi et al., 2020).

SDN has emerged as an excellent networking choice for managing the complex QoS network infrastructure within the cloud. A key requirement of the networking infrastructure is to provide adequate cybersecurity and protection for cloud services without compromising QoS or service openness.

SDN can enhance and automate cloud data center operations by providing numerous benefits to cloud service providers, including:

(1) Cost-saving: SDN architecture is not expensive, and the SDN controller can support layer 1 through layer 3 network devices.
(2) Intelligent framework: an SDN controller can develop an intelligent and globally connected environment by managing critical issues like load-balancing cloud data center infrastructure by routing network traffic to appropriate data centers based on the network load.
(3) Network automation: SDN controller can manage data center traffic flow, lowering downtime and increasing data resilience. SDN can automatically allocate more cloud resources to support increasing network loads, which is a significant benefit for users, as they never worry about the scalability and availability of their resources as long as the SDN controller can manage the cloud platform efficiently.
(4) Network security: an SDN controller acts as a centralized control point for uniformly disseminating security and policy information across the enterprise. SDN enables administrators to dynamically apply security policies to network functions, with tight integration between security, data, applications, services, and the network to ensure that network users and network services are authorized and that all proper policy limitations have been imposed.

The architecture of the SDN is divided into three components (Haleplidis et al., 2015) as displayed in Fig. 2. (a) The infrastructure layer contains the network devices that forward the network traffic based on the rules installed by the centralized SDN controller. (b) The control layer is a software program running on a computer server that acts as the brain of the entire network that can monitor all devices and manage them by installing the forwarding rules centrally. The controller monitors the overall status of the network and manages the

![Fig. 2. Layered architecture of SDN network (El-Shamy et al., 2021).](image-url)
relationship between network infrastructure resources, management policies, and application delivery. The controller periodically updates the flow table with appropriate management rules and collects flow statistics. (c) The application layer communicates with the controller through the northbound API interface to exchange information and instructions to deploy network policy and manage network traffic (El-Shamy et al., 2021).

Layered architecture of the SDN (El-Shamy et al., 2021).

Although the SDN security framework is well designed, there are still some security challenges to be addressed, and attacks can have a severe impact on all components of SDN. Some of these threats are common with traditional network architectures, while others are unique to the SDN. There are two types of security threats targeting the SDN architecture. (a) SDN-specific attacks target the controller itself, the communication channel between the controller and the network devices, or the communication channel between the controller and the running applications. (b) Common attacks between the traditional network architecture and SDN target the application layer and network devices. Security threats targeting the SDN architecture can be classified into the following four vectors (Deb and Roy, 2022).

(1) Attacks on the data plane: the attacker can target the network devices in the data plane in many ways by sending malicious traffic or flooding the network to consume the available space in the flow table in SDN switches besides modifying the flow table rules to reroute the network traffic to illegitimate users; also, deployment of a bogus virtual switch to steal data; and finally, sending a lot of fake requests to the controller to overwhelm the controller and waste its resources or to consume the network bandwidth.

(2) Attacks on the application layer: the attacker may run malware or start a malicious application that can deceive the SDN controller to modify the flow table entries in the network devices to violate the security policy.

(3) Attacks on the SDN controller: the SDN controller is the main component of the entire network and acts as the brain of the DC network. Controlling access to the controller can cause the entire system to collapse. In addition, the attacker can deploy a fake controller within the network and install its own rules on network devices to disrupt the traffic forwarding process. Furthermore, the attackers can exploit vulnerabilities in the controller's operating system.

(4) Attacks on the communication channel between the SDN controller and the network devices: the controller manages the network devices through a logically separate channel for every network device. However, these channels share the same physical network links. An intruder can launch an attack on those channels by transmitting a flood of packets that causes network congestion.
As a result, the controller becomes isolated from the entire network. In addition, the attacker can target the trust relationship between the controller and the network devices to launch a man-in-the-middle attack and capture the transmitted data.

2.2. Intrusion detection system

IDS is a powerful security solution that monitors network traffic for suspicious activities; whenever malicious activity is detected, it sends an alert to the SDN controller and network administrator to start prevention measures. Also, these events are collected and stored centrally in security information and event management systems for further analysis. IDS can be classified into various types as shown in Fig. 3 (Vaneeta, 2021).

Classification of IDS

IDS can be classified according to the data source into host-based IDS (HIDS) and network-based IDS. (a) HIDS is a software application running on end hosts on the network to monitor incoming and outgoing packets from only this device. HIDS can detect attacks with a high degree of accuracy. However, HIDS must be installed on every host, and it cannot integrate with the existing network security solutions. (b) Network-based IDS is a hardware device setup within the network to analyze network traffic, and integrate easily with existing security solutions such as firewalls.

IDS can be categorized according to the detection technique into signature-based and anomaly based. (a) The signature-based detection method discovers malicious traffic by matching the incoming traffic with its database of known attacks to decide the nature of this traffic. These approaches have proved efficient for detecting known attacks. However, signature-based IDS cannot detect unknown or new threats whose signature is not found in the IDS database. So, the IDS database must be updated periodically with the latest attack signatures. (b) Anomaly-based detection methods monitor the behavior of the network traffic to build a profile of the normal network behavior, and in the case of any new incoming traffic that deviates from this normal behavior, the model will define it as an anomaly. ML techniques are widely used in this method by training a security model to classify anomalous activities from benign traffic. Generally, anomaly-based detection techniques are more powerful than signature-based as they can detect new and unknown attacks, but signature-based methods have higher accuracy.

IDS is classified into passive and active IDS based on the action taken. (a) Passive IDS simply monitors and analyzes network traffic, alerting the administrator...
or SDN controller if vulnerabilities are detected. (b) Active IDS is an IDPS. It performs all the IDS functions as well as responds to the threat by blocking the attacker’s traffic. Finally, IDS can be classified according to the working mode into offline IDS and online IDS. (a) Offline IDS is not a real-time IDS as it analyzes the events long after the attack. (b) Online IDS analyzes the network traffic in real time.

2.3. Ensemble learning

Ensemble learning can be classified according to the base classifier learning method into boosting and bagging as shown in Fig. 4 (González et al., 2020).

(1) Boosting technique converts weak classifiers into strong classifiers in a forward stagewise process to decrease the classifier bias problem (underfitting) by increasing the weights of misclassified samples in each successive iteration. The boosting algorithm calculates the final result by combining the output of all iterations using a weighted vote for classification problems or a weighted sum for regression problems. Boosting ensemble algorithms that are commonly used include adaptive boosting (AdaBoost), gradient boosting machine (GBM), categorical boosting (CatBoost), and extreme gradient boosting (XGBoost), and LightGBM (Wang and Sun, 2021).

(2) Bagging technique combines a set of independent base classifiers and trains them on different subsamples of the original data to overcome the problem of high variance (overfitting). To implement a bagging model, multiple datasets are generated by bootstrapping the training data, after which models are developed based on the individual datasets and predictions are made using these models. All the prediction results are merged to generate a final value using mathematical methods such as the majority vote in classification problems and averaging in regression cases. Because an individual learner is often sensitive to noise in the training data, bagging should provide stable and improved results with lower variance by aggregating multiple results into a single prediction (Wang and Sun, 2021).

The nature of network traffic has skewed class distributions and is prone to significant bias. Hence, it is natural to use the boosting methods over bagging to handle the class imbalance problem. AdaBoost is an ensemble learning method, and there are many variants that have been developed with different characteristics and purposes. The main idea of AdaBoost is to use an iterative approach to learn from the mistakes of weak classifiers and turn them into strong ones. In every iteration, the base classifier is trained using the weights of the training samples and is updated according to the performance of the past iterations. The weights are calculated based on the used AdaBoost versions (Wang and Sun, 2021).

After each AdaBoost iteration, the algorithm updates the dataset sample weight according to the classification errors. It increases the weight value of the misclassified instances and reduces the weight of correctly classified samples. This iteration is continued until the distribution of the weighted samples of misclassified and correctly classified samples is equal. However, it increases the weight of misclassified instances of all classes by an equal ratio; in addition, it reduces the weight of correctly classified instances of all classes to the same percentage. The optimal boosting strategy should be able to distinguish between different classes and assign higher weights to classes with few samples, which are usually misclassified to overcome the imbalanced class problem and improve the detection performance of rare attack samples (Yan and Han, 2018).

The workflow of bagging and boosting techniques (González et al., 2020).

Cost-sensitive learning integrates with adaptive boosting for further improvement of classification accuracy, as it adds a penalty or misclassification cost to the predictions that lead to the wrong classification based on the main goal of minimizing the entire cost of AdaBoost ensemble learning. The cost-sensitive algorithm calculates a cost matrix, which is used as an additional input for the base classifier during the learning sequences and also to evaluate the base classifier’s performance in reducing misclassification costs. The cost function updates the weights after each iteration (Yan and Han, 2018).

3. Related work

This part provides an overview of network intrusion solutions to detect rare attacks that have been studied over the last decade, as explained in the following section.

3.1. Intrusion detection approaches ignoring class imbalance problems

Several ML techniques have been used to design an effective IDS. However, these methods do not consider the imbalanced classes in the dataset and
try to improve the overall detection rate without considering the performance of the detection rate of rare attacks. This section discusses the related work for IDS solutions on SDN that use various ML and deep learning techniques.

Choobdar et al. (2021) proposed an IDS based on SDN that consists of three phases. (a) The pre-training phase uses sparse stacked auto-encoders for feature selection. (b) The training phase uses the softmax algorithm to train the model. (c) The tuning and optimizing the system parameters phase. The authors evaluated the performance of the proposed system using NSL-KDD and CSE–CIC–IDS2017 datasets. The accuracy of attack detection and classification was promising compared with previous approaches. Tang et al. (2020) proposed an IDS method in the SDN architecture using two deep learning methods: Deep Neural Network (DNN) and Gated Recurrent Neural Network (GRU-RNN). They evaluated their model using the NSL-KDD dataset. Ibrahim and Bhaya (Ibrahim and Bhaya, 2021) proposed an IDS solution for cloud-based SDN using support vector machine (SVM) with GridSearch to detect attacks. The authors used the UNSW-NB-15 and NSL-KDD datasets to train and test the proposed method. They have proven that their technique is effective against attacks on SDN-based cloud environments.

Al-Qatf et al. (2018) proposed an IDS based on deep learning with the primary objective of reducing training time and optimizing accuracy performance. They used a sparse auto-encoder for feature selection and the SVM algorithm to identify attacks. The authors evaluated their method using the NSL-KDD dataset. Taher et al. (2019) introduced an artificial neural network method for their IDS solution, focusing on enhancing the detection rates. They evaluated the proposed method's performance using the NSL-KDD dataset, where they found that the proposed method outperformed other methods. Gao et al. (2019) suggested an adaptive ensemble ML model that uses various shallow ML algorithms like SVM, RF, k-nearest neighbors, decision tree, and logical regression. The proposed method aims to overcome the low performance of a single shallow ML algorithm to increase the detection rate of attacks. They tested the effectiveness of the proposed solution using the NSL-KDD dataset. They proved that their approach improves detection accuracy.

However, all these six approaches evaluated their solutions using the NSL-KDD dataset, which is a legacy dataset and does not contain recent or a variety of attacks that limit their methods for detecting current attacks.

Yang and Zhao (Yang and Zhao, 2018) proposed an IDS method in SDN to detect and mitigate DDoS using the SVM. The authors used the NSL-KDD dataset, where they found that the proposed hierarchical IDS scheme has a high detection rate with low false positive and error rate. However, the packet-level solutions proposed in the study by Schueller et al. (2018) can easily be evaded by injecting noise from attackers. Therefore, the study by Schueller et al. (2018) can easily be evaded by injecting noise from attackers.
those solutions always fail to detect malicious traffic when attackers conduct such an evasion method. Meanwhile, those packet-level solutions also failed to detect 0-day attacks accurately for the weak robustness. Moreover, the state-of-the-art methods based on traditional rule methods, which analyzed numerous statistical traffic features, always introduce significant detection latency. Furthermore, the DARPA dataset was created with an out-of-date version of the TCP protocol. According to modern standards, using the old TCP version makes the header field ‘IPv4 Type of Service (ToS)’ invalid.

Several binary IDS solutions based on SDN have been proposed to discover attacks without identifying their type. Tang et al. (2018) proposed an IDS model for SDN using a DGRU-RNN with three hidden layers to perform binary classification. They trained the model with six basic feature selections from the NSL-KDD dataset, which enabled the intrusion detection model and achieved a high degree of accuracy. Isa and Mhamdi (Isa and Mhamdi, 2020) proposed an IDS for the native SDN using a hybrid unsupervised ML approach based on auto-encoding and a random forest algorithm. Vargas-Munoz et al. (2018) used a Bayesian network to detect worms and brute-force attacks, while Hamed et al. (2018) used the bigram technique to detect stealthy attacks. Gurung et al. (2019) proposed an IDS based on deep learning that used a sparse auto-encoder (AE) to learn features and then logistic classification to detect attacks.

However, given the great diversity of security threats and the considerable increase in network data, it is essential to determine the type of attack to take appropriate action and mitigate these threats according to their kind. In addition, Tang et al. (2018) used only six out of 41 features when deploying the NSL-KDD dataset, which is reflected in the performance of the classifier algorithm and indicates a low detection rate as the used subset of features is not sufficient to detect malicious traffic.

Yang et al. (2020) built an IDS model within the SDN controller, which is an ensemble of multiple layers with an implementation auto-encoder method (Griffin), to improve the anomaly detection accuracy. The authors tested their proposed model using Yisroel Mirsky’s dataset. Fausto et al. (2022) proposed a novel IDS solution for SDN to collect and analyze network traffic. The authors suggested a probability-based traffic sampling method using the genetic algorithm to avoid network congestion due to the large amount of data transferred between network devices and the SDN controller. Based on the simulation results, the suggested model can enhance the detection efficiency in an SDN environment.

However, the proposed IDS solution in Yang et al. (2020) was evaluated using a nonstandard security dataset that did not contain a large number of the collected network traffic samples and features, making the obtained results questionable and suspicious. In addition, the suggested IDS algorithm in Fausto et al. (2022) is installed on the SDN controller instead of a stand-alone server, which adds extra load to the controller and can degrade network performance, especially in a large network consisting of massive network devices with heavy loads of network traffic.

Furthermore, these previous IDS approaches have several drawbacks. It is common knowledge that class imbalance occurs in intrusion detection datasets, although many studies ignore it. Also, rather than following a systematic approach, the size of the training sample is typically chosen at random. They are also constrained by the use of out-of-date datasets like KDD-CUP-1999, NSL-KDD, or DARPA datasets.

3.2. Intrusion detection approaches handling class imbalance problems

Many previous IDS solutions have been proposed to overcome the classification problem of imbalanced data. These approaches can be classified into data preprocessing and algorithmic techniques as explained in Fig. 5. The following section discusses previous studies based on the following two approaches.

Imbalanced dataset handling methods (Rout et al., 2017).

Data-level solutions are carried out on the data before the classification process to rebalance the classes of the dataset. They include undersampling and oversampling methods. As shown in Fig. 6, undersampling methods delete some samples from the majority class, whereas oversampling methods increase the number of instances within the minority classes by creating new synthetic data samples (Rout et al., 2017).

Data sampling techniques (Rout et al., 2017).

Xiao and Xiao (Xiao and Xiao, 2019) proposed IDS using Simplified Residual Networks, a modified architecture of residual networks. They balanced the NSL-KDD dataset using the random oversampling technique by duplicating minority class samples randomly. However, this approach replicates the training data samples exactly, which may lead to overfitting as the classifier is trained continuously using the same data.
Jeatrakul et al. (2010) proposed an oversampling technique using the synthetic minority oversampling technique (SMOTE) and then using the complementary neural network classification to detect attacks. They tested the proposed method using the UCI dataset. The results indicate that the proposed approach effectively handles the class imbalance problem. Karatas et al. (2020) compared the performances of RF, AdaBoost, k-nearest neighbors, gradient boosting, decision tree, and linear discriminant to detect intrusion. They evaluated these various six classifiers using CSE–CIC–IDS2018 after handling the class imbalance using the SMOTE technique. The experimental results showed that the AdaBoost classifier achieves the highest detection accuracy for rare intrusions. Anchuri et al. (2022) proposed an improved IDS model for SDN based on modified deep forest classification using a multistage hybrid and intelligent deep learning classifier. The authors handled the imbalance issue in the dataset using SMOTE. They evaluated the proposed method using the NSL-KDD dataset, where the experimental results demonstrated an improvement in the performance accuracy of the proposed method.

However, SMOTE technique used in these methods faces the problem of overgeneralization. It did not consider the majority class as it blindly generalized the region of a minority class. That will lead to a greater chance of class mixture if the class distribution is highly skewed.

Yan and Han (Yan et al., 2018) proposed an IDS method based on deep learning to overcome the overgeneralization issue. They handled the data imbalance in network traffic using the local adaptive SMOTE, a modified version of SMOTE. They used the GRU DNN to identify network attacks. Yao et al. (2021) proposed an IDS for SDN using feature engineering-based autoencoder and LightGBM. The proposed system first uses borderline-SMOTE to optimize data distribution, then AE is used for feature engineering to extract key features. Finally, LightGBM is trained to detect attacks using extracted features. The proposed method has been tested on the KDDCup99 and NSL-KDD datasets. Experimental results on the KDDCup99 and NSL-KDD datasets show that the accuracy, precision, and F1-score performance of the proposed model are better than those of traditional models and related works.

Kurochkin and Volkov (Kurochkin II & Volkov, 2020) proposed an IDS solution for SDN-based networks using a GRU-RNN model to detect and classify the attack. They used a hybrid sampling algorithm to address the class imbalance problem by undersampling the benign traffic and oversampling the attack samples using the SMOTE technique.
technique. The authors evaluated their proposed solution using the CIC-IDS-2018 dataset. Although the model achieved an F1-score for detecting DDoS attacks, it yields a low detection score for web and infiltration-labeled attacks.

The drawback of hybrid sampling is that the computation cost of a classification model is increased. However, any resampling method may introduce a bias into the classification process due to the change of inputs and attack distribution. For instance, the new synthetic data might be redundant, whereas removing valuable data might lead to poor performance.

Xuan-Qui et al. (2021) proposed an IDS solution for the SDN-based network using a generative adversarial network (GAN) to overcome the imbalance in the network traffic. The proposed GAN framework generates malicious data streams that mutate from real attack traffic and are used to train the IDS to improve the robustness of the IDS in detecting sophisticated attacks. The experiments are performed on the CICIDS2017 dataset and evaluated using two performance metrics: detection rate and F1-score (F1).

However, GAN-based solutions are facing two major problems. (a) Mode collapse: the generator collapses as it produces limited varieties of samples. (b) Nonconvergence: the model parameters oscillate, destabilize, and never converge.

Le and Thinh (Le and Thinh, 2021) proposed an IDS solution for SDN using a combination of ML techniques, including: (a) a deep sparse autoencoder for reducing the dataset dimension and feature selection. (b) A conditional generative adversarial network (CGAN) for addressing the dataset imbalance problem. (c) Ensemble learning techniques to detect security attacks. The simulation results demonstrated that the proposed approach is capable of fast network traffic classification with a high detection accuracy rate and a relatively low false alarm rate. Although the CGAN model can be trained to solve a variety of problems and has overcome the drawbacks of a GAN-based model, namely mode collapse and nonconvergence, the CGAN-based solutions lack generalizability because they are a much less reliable method when applied to data with a slightly different structure than the training data. This presents a major disadvantage when applying CGANs to scientific problems, where the accuracy of solutions is critical.

The algorithmic methods change and modify the classifier algorithms to consider different class distributions and adapt them toward the minority classes. It includes two approaches: cost-sensitive and ensemble learning techniques.

Lin and Hongle (Lin and Hongle, 2020) proposed an algorithm-level IDS model for SDN based on an online ensemble learning algorithm. The bagging algorithm is used to minimize the impact of data stream imbalance on the performance of the integrated classifier through the improvement of unbalanced data detection, a dynamic penalty factor, and selection integration. The authors used the NSL-KDD dataset to simulate the data stream in the SDN, and the experimental results demonstrated that the proposed model can enhance the detection accuracy, especially the detection rate of unknown intrusion attacks.

Bedi et al. (2020) proposed an algorithm-level IDS method based on the siamese neural network. The proposed model can detect rare attacks without using sampling techniques. They compared the performance of the proposed model against that of DNN and CNN. The result of the proposed model has a higher recall value for rare attacks compared with the other two algorithms. The proposed model obtained good results for the recall values, but the precision values were low for the minority classes.

Cost-sensitive learning techniques assign a distinct misclassification cost value for each class within the dataset. Typically, it tries to reduce the total misclassification cost by setting a high-cost value for the minority class and a small cost value for the majority class.

At the cost-sensitive level, Chao et al. (2019) proposed an online IDS security strategy in an SDN environment to classify and identify network traffic based on stream mining. The proposed algorithm is built in two phases. (a) The first classification phase is deployed at the network switches. It is a decision tree-based, cost-sensitive classification algorithm. (b) The second phase is deployed at the SDN controller. It is a Hoeffding tree-based stream mining algorithm. It performs incremental learning on the first five packets of each flow, which guarantees a short classification delay. Although the proposed model handles data imbalance and online classification problems differently than most of the existing network traffic detection and classification approaches, it has a few limitations. The detection results of the second phase depend completely on the results of the first phase, which used two network traffic features only as training features, namely the destination port number and the protocol, which may bias classification performances. Also, the paper does not discuss the impact of phase one classification on the real-time aspect of the classification. Sharif and Moein (Sharif and Moein, 2021) proposed a convolutional neural network to classify traffic and handle the imbalanced datasets.
issue using the cost-sensitive technique by assigning a cost value to the minority classes higher than the majority classes in the loss function of CNN. They evaluated their model using the ISCX_VPN-non-VPN dataset (Tong et al., 2018), where the results demonstrated the superiority of their proposed approach over other traffic classification techniques. Blanco et al. (2018) proposed an IDS based on reinforcement learning to build a cost-sensitive classifier using the DQN architecture over MLP.

However, the DQN adds additional effort during the training phase. They evaluated their approach using NSL-KDD and UNSW datasets, where the results reduced the false-negative errors by changing the rewarding strategies. However, the critical issue with the above is that they did not consider the cost of different classes and how to define proper misclassification of cost values for multiple classes.

4. Proposed system architecture

The proposed IDS solution integrates with the SDN controller to detect malicious network activities and take appropriate countermeasures to prevent their harmful effects on network resources. The controller monitors the network and sends the collected data to the IDS to analyze the network data and notify the controller whenever a threat is detected. Based on this notification, the controller prevents this flow to protect the DC.

The proposed model consists of three modules, as depicted in Fig. 7. The network traffic monitoring module collects traffic statistics from switches using the OpenFlow protocol (Le and Thinh, 2021). The analyzing module detects attacks and security threats using ensemble learning. Flow management modules install flow rules on switches to route network traffic to a destination or block harmful attacks. The following part explains each module in detail.

4.1. Monitoring module

The primary function of the SDN controller is to monitor the entire cloud data center network and collect flow statistics. The SDN controller tracks the existing and new flows using ‘Packet-In’ and ‘STATS’ messages based on the original OpenFlow protocol, as follows.

(1) When a switch receives a new network flow, it checks its flow table to find a matching rule. If there is a rule in the flow table, the switch forwards the flow according to this rule; otherwise, the switch encapsulates the first packet of flow in a Packet-In message and sends it to the controller, which generates flow rules and installs them in the switches’ flow table. Therefore, the controller extracts the flow features from all corresponding Packet-In messages.

(2) The SDN controller periodically requests the switches to send statistics about network traffic using the ‘Stats_Request’ message that queries the network devices about their status, and the second message is the ‘Stats_Reply’ that carries the flow information statistics about the network traffic (Lin and Hongle, 2020).

4.2. Analyzing module

This module processes and analyzes the collected data to detect threats. If it detects any malicious activity, an alert will be issued to inform the controller about the existence of the intrusion and its type. This module has three functions: data preprocessing, cost-matrix optimization, and attack detection. The details of these two functions are as follows.

(1) Data preprocessing: the collected data has different features and characteristics. Therefore, it is recommended to preprocess the collected data to produce a valid data input for the classification algorithm. Data preprocessing is a primary step performed in ML to address dataset defects such as missing values, duplicate values, and errors that can affect classification performance.

The data preprocessing includes the following four steps.

(a) Data cleaning: this step includes dropping the duplicate values, deleting any record containing null or infinitive values, and deleting feature columns with a single constant value. These values increase the overhead and do not enhance the training phase, as they do not play any role in differentiating between categories.

(b) Data numeration converts string values into numeric.

(c) Data normalization: using a logarithmic scaling function to rescale the values in the range from 0 to 1 as in (1).

(d) Feature selection: this step selects the most relevant features and deletes the irrelevant dataset features to reduce the number of redundant and noisy features:

$$x'_i = \frac{x_i - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} \quad (1)$$
where $x'_i$ is the normalized value; $x_i$ represents the selected feature value; $x_{\text{max}}$ is the maximum feature value; and $x_{\text{min}}$ is the minimum feature value.

Block diagram of the proposed algorithm.

(2) Cost-matrix optimization: the proposed algorithm uses grid search and genetic search hyperparameter optimization algorithms to determine the best cost-sensitive weight value for each class in the dataset, as indicated in the following two steps.

Grid search: (a) the proposed algorithm creates an initial random arbitrary C cost vector, each vector containing a K element, where K is the number of classes. (b) Runs the grid search algorithm and calculates the value of the fitness function. This algorithm considers the fitness function for each cost vector to be the geometric average of macro recall (G-mean), as explained in part 5-A. The G-mean is selected as a fitness function because it handles the identification of all classes with equal importance. (c) After trying all combinations, the best vector is selected that has the highest impact on the fitness function based on G-mean values, as shown in Fig. 8. (d) Repeating steps 1–3 for k times to obtain k optimized cost vectors. The grid search output is fed to the genetic algorithm as in the next phase.

Grid search algorithm flow.

The genetic search steps are as follows: (a) The proposed algorithm starts with the optimized cost values generated from the grid search instead of selecting a random cost matrix. (b) Runs the grid search algorithm and calculates the value of the fitness function. This algorithm considers the G-mean as the fitness function, as selected in the grid search. (c) The selection step chooses two cost vectors that have a higher impact on the fitness function based on G-mean values. (d) The crossover step uses the arithmetic average to merge the two selected cost vectors and generate one cost vector. (e) The mutation step adjusts the elements of the cost vector using a random number, as shown in Fig. 9. (f) Repeat steps 2–5 for P times. (g) The optimized cost matrix contains a C vector representing the misclassification weight of each class.

Genetic algorithm flow.

The proposed algorithm selects the grid search due to its simplicity, and the genetic algorithm is
used due to its efficiency in hyper-parameter optimization in the various iteration steps, which are computed based on the previously calculated values (Bedi et al., 2020; Chao et al., 2019).

(3) Attack detection: the proposed method integrates the adaptive boosting (AdaBoost) classification technique with the cost-sensitive methods to enhance the detection process of attacks, with a special focus on rare attacks.

AdaBoost is an iterative algorithm that combines multiple weak classifiers to build a strong classifier to enhance the classification performance. The proposed algorithm uses the decision tree classifier as a base classifier (Sharif and Moein, 2021).

The proposed AdaBoost algorithm is based on Stagewise Additive Modeling using multi-class exponential loss function, which was introduced by Tong et al. (2018).

The following steps illustrate the details of the detection method based on AdaBoost Stagewise Additive Modeling using multi-class exponential loss function, as shown in Fig. 10. Let the collected dataset D containing N samples can be represented as in (2):

\[ D = \{(x_1, y_1, c(y_1)), (x_2, y_2, c(y_2)), \ldots, (x_i, y_i, c(y_i))\} \]  

(2)

where \( x_i \) is a sample in the dataset and each sample has a set of \( m \) features \( F = (f_1, f_2, \ldots, f_m) \), \( y_i \) represents the class label of \( K \) values \( y_i \in \{1, 2, \ldots, K\} \).

The first step is to set the initial weight value for all \( N \) samples in the dataset by the weight function, as in (3):

\[ D_t(j) = (c(1), c(2), \ldots, c(k)) \quad j = 1, 2, 3 \ldots K \]  

(3)

where \( c(j) \) represents the cost value for class \( (j) \), previously calculated using hyper-parameter optimization techniques.

The steps of the rare attack detection module.

The algorithm trains the weak classifier \( h_t(x_i) \) over the dataset \( D_t \) in an iteration cycle and computes the performance of this weak classifier:

\[ h_t(x_i) : X \rightarrow k \in \{1, 2, \ldots, K\}. \]  

(4)

The base classifier’s performance is evaluated by calculating the misclassification errors (\( \in_t \)) using (5):

\[ \in_t = \frac{\sum_{i=1}^{N} D_t(i) I(y_i \neq h_t(x_i))}{\sum_{i=1}^{N} D_t(i)}. \]  

(5)
where $D_t(i)$ is the weight of dataset samples; $h_t(x_i)$ is the base classifier at step ($t$); $t=1,2,\ldots,T$; and $T$ is the number of iterations in ADABoost learning.

Based on the last classification result, the algorithm updates the weight of the samples in the subsequent iteration, where the misclassified samples are given a higher weight.

Throughout the iterative learning phase, the algorithm aims to gradually decrease the final classifier error by assigning a higher weight value to the misclassified samples. The weight function of the base classifier is calculated as in (6):

$$\alpha_t = \log \left( \frac{1 - \varepsilon_t}{\varepsilon_t} \right) + \log (K - 1)$$

The value of $\alpha_t$ represents the contribution of each classifier $h_t(x_i)$ to build the final ensemble classifier $H(x_i)$. As the weighted error $\varepsilon_t$ controls the calculated value of $\alpha_t$, and to avoid any negative values of $\alpha_t$, the condition in equation (7) must hold:

$$(1 - \varepsilon_t)/(K - 1) > \varepsilon_t$$

Therefore, the limits of weighted error values are as in (8):

$$\varepsilon_t < \frac{K - 1}{K}$$

The algorithm updates the sample weight in the dataset using the new calculated value $\alpha_t$ and cost vectors; next, the algorithm normalizes each sample by dividing it with the sum of samples weights as in (9).

These samples are the inputs to the base classifier in the following round of learning:

$$D_{t+1}(i) = \frac{C(y_i) D_t(i) e^{-\alpha_t I(y_i = h_t(x_i))}}{\sum_{j=1}^{N} C(y_j) D_t(j) e^{-\alpha_t I(y_j = h_t(x_j))}}$$

After finishing $T$ training cycle for the base classifiers, the final classifier model is generated as in (10):

$$H(x_i) = \arg\max_k \sum_{t=1}^{T} \alpha_t I(h_t(x_i) = k)$$

**Algorithm 1: the proposed attack detection algorithm**

**Input:** network traffic, base classifier, number of iterations ($T$)
**Output:** build ensemble classifier, detect attacks
1: Calculate the initial weights $D_1(i)$ for all using Eq. (3)
2: Using GA calculate cost vector values ($C$)
3: For $t = 1,2,\ldots,T$ do/*start iterative ensemble for T rounds*/
4: Fit the base classifier with weight instances $D_t(i)$
5: Calculate the base classifier error ($\varepsilon_t$) using Eq. (5)
6: Calculate classifier weight using Eq. (6)
7: Update samples weight using Eq. (9)
8: End for/*finish ($T$) iterative learning steps*/
9: Build final ensemble model using Eq. (10)
10: For each new flow ($f$) of (F)
11: Network traffic arrives at an SDN switch
12: Switch sends the first packet to the controller
13: Controller contacts IDS to check this flow
14: If this network traffic is benign then
15: SDN controller install forwarding rule
16: Else if the network traffic is attacked
17: The IDS detect the attack type
18: SDN controller create a block rule
19: Controller update switches rules
20: Send alert to admin and update the IDS database
21: End if
22: End for
This module is responsible for taking corrective measures to prevent and mitigate harmful attacks. Once the attack is detected, the controller immediately installs and updates flow entries on the tables on switches with the appropriate forwarding rules in flow tables in SDN switches to drop the malicious flow packets. For normal traffic, the controller installs rules in the SDN switches to forward these benign flow packets to their destination.

The following pseudocode describes the proposed algorithm to build an AdaBoost ensemble classifier that can detect attacks, focusing on improving rare attack detection.

Algorithm 1 works as follows. In lines 1–2, the algorithm calculates an initial weight value $D1(i)$ for all samples in the dataset using grid search and genetic search. In lines 3–8, the iterative ensemble learning starts to train the base classifier for ($T$) rounds with data samples and weights for these samples. The algorithm assigns higher cost values for rare attacks to force the classifier to detect them. After a training cycle is finished, the algorithm calculates the error due to misclassified samples. Before starting a new training cycle, the algorithm calculates the new sample weight where the value of the rare attack weight is higher than the benign traffic. This process is continued till the completion of all ($T$) training cycles. In line 8, the final ensemble model is generated, which has more classification accuracy than the base classifier and is trained to detect rare attacks.

In lines 10–22, the SDN controller integrates with IDS to protect the network against attacks, when a new flow arrives switch; it forwards the first packet to the controller. The controller checks with the IDS the types of this packet. If it is normal traffic, then the controller installs a forwarding rule on all switches in the path of the packet till it arrives at its final destination. Else if this packet is detected as an attack, then the controller sends a rule to the network switches to block this packet, sends an alarm to the network administrator, and updates the IDS security database.

This paper uses two recent security benchmark datasets to evaluate the proposed model: CSE–CIC–IDS2017 (Blanco et al., 2018) and CSE–CIC–IDS2018 (Braun and Menth, 2014). The number of records in the CSE–CIC–IDS2017 dataset is 2 830 743 records, while CSE–CIC–IDS2018 contains 16 232 943 records, representing normal traffic and 14 different types of attacks. The number of features in both datasets is 80 features. Table 1 provides detailed information about the two datasets, indicating a severe imbalance between normal data and attack types. The imbalance ratio can be calculated as the ratio between majority and minority classes (Zeng et al., 2016) as in (11):

$$\text{Imbalance Ratio} = \frac{\max(N_c)}{\min(N_c)}$$  

where $\max(N_c)$ represents the total sample number of the majority class, and $\min(N_c)$ is the total sample number of the minority class.

### 5.1. Security datasets

#### 5.1.1. CSE–CIC–IDS2017 dataset preprocessing

The steps of preprocessing CSE–CIC–IDS2017 before using it to train the algorithm are as follows.

1. Concatenate the eight files of the CSE–CIC–IDS2017 dataset into one file.
2. Delete 1358 records with missing values and delete 1509 records with infinite values.
3. Drop the column ‘Fwd_Header_Length.1’ as it is duplicated of column ‘Fwd_Header_Length’.
4. Drop column with fixed constant values, these columns are: ‘Fwd_PSH_Flag,’ ‘Fwd_URG_Flag,’ ‘Fwd_Avg_Bytes/Bulk,’ ‘Fwd_Avg_Packets/Bulk,’ ‘Fwd_Avg_Bulk_Rate,’ ‘Fwd_Avg_Bulk’.
5. Encode the label column using the Label_Encoder function.
7. The feature selection step selects the most effective feature using the RF feature importance technique. The algorithm selects the first 35 features because their importance decreases after that, and therefore their participation in the classification process is very small, as depicted in Fig. 11. These features are stated in Table 2.
The same preprocessed steps are repeated for the CSE-CIC-IDS2018 dataset.

1. Concatenate the 10 files of the CSE-CIC-IDS2018 dataset in one file.
2. Deleting 59,721 records with missing values and deleting 33,167 records with infinite values.
3. Drop column with fixed constant values, these columns are: ‘Bwd_PSH_Flags,’ ‘Fwd_PSH_Flags,’ ‘Bwd_URG_Flags,’ ‘CWE_Flag_Count,’ ‘Fwd_Byts/b_Avg,’ ‘Fwd_Pkts/b_Avg,’ ‘Fwd_Blk_Rate_Avg,’ ‘Bwd_Byts/b_Avg,’ ‘Bwd_Pkts/b_Avg,’ ‘Bwd_Blk_Rate_Avg’]. These column contain only zero value.
4. Encoding the label column using the Label-Encoder function.
5. Data normalization using minimum—maximum function.

Table 1. Attack class distribution.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Record count</td>
<td>Percentage</td>
</tr>
<tr>
<td>Benign</td>
<td>2 273 097</td>
<td>80.3004</td>
</tr>
<tr>
<td>DoS Hulk</td>
<td>231 073</td>
<td>8.1630</td>
</tr>
<tr>
<td>PortScan</td>
<td>158 930</td>
<td>5.6144</td>
</tr>
<tr>
<td>DDoS</td>
<td>128 027</td>
<td>4.5227</td>
</tr>
<tr>
<td>DoS GoldenEye</td>
<td>10 293</td>
<td>0.3636</td>
</tr>
<tr>
<td>FTP-Patator</td>
<td>7938</td>
<td>0.2804</td>
</tr>
<tr>
<td>SSH-Patator</td>
<td>5897</td>
<td>0.2083</td>
</tr>
<tr>
<td>DoS slowloris</td>
<td>5796</td>
<td>0.2048</td>
</tr>
<tr>
<td>DoS Slowhttp</td>
<td>5499</td>
<td>0.1943</td>
</tr>
<tr>
<td>Bot</td>
<td>1966</td>
<td>0.0695</td>
</tr>
<tr>
<td>Web attack-brute force</td>
<td>1507</td>
<td>0.0532</td>
</tr>
<tr>
<td>Web attack-XSS</td>
<td>652</td>
<td>0.0230</td>
</tr>
<tr>
<td>Infiltration</td>
<td>36</td>
<td>0.0013</td>
</tr>
<tr>
<td>Web attack-SQL injection</td>
<td>21</td>
<td>0.0007</td>
</tr>
<tr>
<td>Heartbleed</td>
<td>11</td>
<td>0.0004</td>
</tr>
</tbody>
</table>

Fig. 11. CSE-CIC-IDS2017 features importance.
As depicted in Figs. 11 and 12, the first 35 features have great importance. After that, the feature’s importance value becomes very small and greatly decreases. So, to optimize the classification speed, the proposed algorithm selects the highest 35 features, as stated in Table 2.

CSE—CIC—IDS2017 features importance.
CSE—CIC—IDS2018 features importance.

5.2. Performance metrics

Choosing an appropriate performance metric for a highly imbalanced dataset classification problem is an important issue. This paper handles imbalanced multiclass classification problems, so this paper uses more than one performance metric, macroaveraged F1-score and G-mean, to better evaluate the results of the ML algorithms with the highly imbalanced dataset. These selected two metrics are the most appropriate metrics for evaluating the imbalanced datasets for the following reasons (Liashchynskyi and Liashchynskyi, 1912).

(1) Most of the standard metrics, such as classification accuracy or classification error, assume a balanced class distribution; actually, the classes are not equal in all security datasets. However, accuracy is not an accurate indicator of the classification performance of imbalanced data, where the minority class has a small number of samples compared with the majority class (Liashchynskyi and Liashchynskyi, 1912). For example, if an imbalanced dataset consists of two classes with a ratio of 98 : 2 and the classification accuracy of the used ML algorithm is 98%, which is misleading and meaningless as it cannot identify any rare class samples. This value is dangerously misleading and becomes unreliable when assessing an imbalanced dataset as it simply detects all instances as being of the majority class. Then this classification accuracy percentage is accepted as the effect of minority classes on overall accuracy is very small. But this result is deceptive and hides the true performance of the model, which is objectively not good because the minority samples are totally missing and the ML algorithm failed to identify these attacks as it only predicts for one class, which is benign traffic (Liashchynskyi and Liashchynskyi, 1912).

(2) The area under the receiver-operating characteristic curve (ROC-AUC) metric is another important metric widely used to evaluate many classification algorithms. However, it is not appropriate for evaluating an imbalanced dataset because an imbalanced dataset with few samples of the minority class, the ROC-AUC, can be misleading. This is because a small number of correct or incorrect classifications can result in a large change in the ROC curve or ROC-AUC score (Liashchynskyi and Liashchynskyi, 1912).

(3) Therefore, this paper uses F1-score and the geometric average of recall (G-mean) metrics as good indicators for the proposed classifier’s performance when handling an imbalanced dataset. The algorithm calculates precision (P), recall (R), F1-score, and G-mean separately on a class basis. Then the algorithm calculates the macro average values of F1-score and G-mean by adding them all together and dividing them by the sum of the number of classes. The macro F1-score represents the harmonic mean of precision and recall for a class $i$. A higher macro F1-
score value is the target for a high-performance classifier. The G-mean is calculated using various macro-recall values for all classes in the dataset. The recall is typically a more attractive metric for an imbalanced classification measurement, as the value of the G-mean will be low when a minority class has a small recall, and if the classifier does not recognize a minority class (recall = 0), then the G-mean = 0. Therefore, a high G-mean value indicates that the classifier has tuned the detection between different classes, and the minority classes get more attention for detection. Therefore, F1-score and G-mean are the ideal metrics to assess the classification performance of imbalanced datasets.

Calculation of these metrics is derived from the confusion matrix.

The confusion matrix contains four elements: true positive (TP), false negative (FN), false positive (FP), and true negative (TN) as shown in Table 3.

TP is the total number of samples predicted as normal, while they were actually normal. FN is the total number of samples classified as normal, while they were actually attacked. However, FP is the total number of samples predicted as an attack, while they were actually normal. Finally, TN is the total number of samples classified as an attack, while they were actually an attack. Where TP is the number of detected attacks, FN is the number of missed attacks, FP and TN.

The proposed algorithm uses a macro-average for P, R, and F1-score as it gives equal weight to each class.

Macro precision: precision (P) is the ratio of the successfully predicted attacks for a class to the total number of attacks. The higher the precision value, the lower the false alarm rate. The precision \( P_i \) is calculated as in (12), while the macro value of precision is the average of precision over all the classes as in (13):

\[
P_i = \frac{TP_i}{TP_i + FP_i} \quad (12)
\]

\[
P_{macro} = \frac{\sum_{i=1}^{K} P_i}{K} \quad (13)
\]

Macro recall: recall (R) is the detection rate that measures the detection accuracy for a class by calculating the portion of attacks in a class (i) that have been correctly detected to the total number of attacks in the system as in (14), while the macro value of recall is the average of recall over all the classes as in (15):

\[
R_i = \frac{TP_i}{TP_i + FN_i} \quad (14)
\]

\[
R_{macro} = \frac{\sum_{i=1}^{K} R_i}{K} \quad (15)
\]

![Fig. 12. CSE-CIC-IDS2018 features importance.](image)
\[ R_i = \frac{TP_i}{TP_i + FN_i} \]  

(14)

\[ R_{\text{macro}} = \frac{\sum_{i=1}^{K} R_i}{K} \]  

(15)

Macro F1-score: the F1-score represents the harmonic mean of precision and recall for a class \((i)\) and is calculated as in (16), while the macro F1-score measure is calculated as the average of F1-score for all classes as in (17). A higher F1-score value is the target for a high-performance classifier.

\[ F_i = \frac{2P_iR_i}{P_i + R_i} \]  

(16)

\[ F_{1\text{macro}} = \frac{\sum_{i=1}^{K} F_1}{K} \]  

(17)

The geometric average of recall (G-mean) is calculated using various macro recall values as in (18). Recall is typically a more attractive metric for an imbalanced classification measurement. The value of G-mean will be low when a minority class has a small recall, and if the classifier does not recognize a minority class (recall = 0), then the G-mean = 0. Therefore, a high G-mean value indicates that the classifier has tuned the detection between different classes, and the minority classes get more attention for detection. Consequently, G-mean is an ideal metric to measure classification performance for imbalanced datasets:

\[ G - \text{mean} = (R_1 \times R_2 \times \ldots \times R_K)^{\frac{1}{K}} \]  

(18)

6. Results and discussion

The various classification algorithms mentioned in this paper are performed using Python 3.9 installed on a workstation running the Ubuntu 20.04.4 LTS operating system with a hardware specification of CPU i7-11800H, memory 32 GB, hard disk 512 GB SSD, and a GeForce GTX 1060 6 GB GDDR graphics card. The two datasets are divided into training and testing with a ratio of 80 : 20.

Note: the full code implementation for all experiments in this paper using the two datasets CSE–CIC–IDS2017 and CSE–CIC–IDS2018 can be found on the GitHub website (https://github.com/ahshamy1/ID).

6.1. Cost vector

The algorithm determines the optimal cost vector in two steps using grid search and genetic algorithm.

First, the grid search algorithm selects the items of the cost matrix randomly based on the imbalance ratio; the cost matrix consists of a number of vectors.

The cost vector contains a 15 cost value for each class in the dataset. The rare attack is assigned a higher cost than benign data. The values of the cost vector are selected in the range from 0.01 to 1.

The algorithm runs many iterations of grid search, wherein each iteration the grid search generates the best vector according to the fitness function (G-mean).

After collecting the best vector in g iteration, the genetic algorithm starts with P population.

The genetic algorithm starts with P population selected from the output of the grid search. The cost vector is optimized using the genetic algorithm.

The proposed algorithm tries various values for the cost matrix using \(P = 6, 8, 10, 12,\) and 14 vectors in the cost matrix. Also, the GA process is repeated using \(I = 5, 10, 15, 20,\) and 25 times (Fig. 13).

The impact of varying cost-matrix size with P vectors and repeating GA for I times on the G-mean value.

As shown in Fig. 14, when the algorithm tries to optimize the cost-matrix size and the number of iterations of the genetic algorithm using different execution rounds of the algorithm, it is found that when \(P = 12\) and \(I = 20,\) values of the cost vectors for each class achieve the optimized G-mean metrics. Although, the values of the cost vector size \(P = 14\) and GA repeating times \(I = 25\) achieve a higher value of the G-mean as depicted in Fig. 14.

However, this enhancement needs a high computational cost and long simulation execution times; thus, the algorithm chose \(P = 12\) and \(I = 20\) values, which achieve higher G-mean values at a reasonable computational cost. These two steps are repeated until the algorithm obtains the optimized cost vector elements for each class in the dataset as shown in Table 4.

Fig. 13. The impact of varying cost matrix size with P vectors and repeating GA for I times on the G-mean value.
model many times with different numbers of iterations: 50, 75, 100, 125, 150, 175, 200, 225, and 250 iterations on the two datasets. Fig. 14 shows the resulting G-mean value for the proposed algorithm using the CSE–CIC–IDS2017 and CSE–CIC–IDS2018 datasets, respectively.

The algorithm increases the number of AdaBoost iterations and calculates the G-mean value. Effect of AdaBoost iteration on G-mean.

To determine the optimal value of AdaBoost iterations that will be used in the algorithm, Fig. 14 shows the G-mean values versus the varying number of AdaBoost iterations. The G-mean value improves as the number of AdaBoost iterations increases. However, the graph shows that after the 150th iteration, the change in G-mean is slight. As a result, for the subsequent tests, the algorithm will use 150 AdaBoost iterations to guarantee an acceptable G-mean value while reducing the computational cost.

Table 5 summarizes the performance metrics against various AdaBoost iterations.

The results of various AdaBoost iterations show that the algorithm obtains better values for the macro F1-score and G-mean as the number of AdaBoost iterations increases. However, the performance is optimized and nearly stable at the 150th AdaBoost iteration, with a small enhancement in the macro F1-score and G-mean when increasing AdaBoost iterations for both datasets. This paper selects the AdaBoost iteration value as 150, which provides reasonable performance considering the high performance, computational cost, and the execution time. The algorithm’s performance improves slightly with the CSE–CIC–IDS2018 dataset over the CSE–CIC–IDS2017 dataset because of its larger sample size.

### 6.3. Decision tree depth

As the proposed AdaBoost ensemble algorithm depends on the decision tree as the base classifier, to

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Macro F1</td>
<td>G-mean</td>
</tr>
<tr>
<td>25</td>
<td>0.79</td>
<td>0.6</td>
</tr>
<tr>
<td>50</td>
<td>0.85</td>
<td>0.73</td>
</tr>
<tr>
<td>75</td>
<td>0.9</td>
<td>0.78</td>
</tr>
<tr>
<td>100</td>
<td>0.953</td>
<td>0.964</td>
</tr>
<tr>
<td>125</td>
<td>0.962</td>
<td>0.966</td>
</tr>
<tr>
<td>150</td>
<td>0.981</td>
<td>0.982</td>
</tr>
<tr>
<td>175</td>
<td>0.981</td>
<td>0.982</td>
</tr>
<tr>
<td>200</td>
<td>0.982</td>
<td>0.982</td>
</tr>
<tr>
<td>225</td>
<td>0.983</td>
<td>0.983</td>
</tr>
<tr>
<td>250</td>
<td>0.984</td>
<td>0.983</td>
</tr>
</tbody>
</table>

These two steps are repeated till the algorithm gets the optimized elements of the cost vector for each class in the dataset as in Table 4.

As depicted in Table 4, the two optimization algorithms, grid search and genetic algorithm, calculate the appropriate weight values of the cost matrix, which achieve the highest G-mean value and reduce the total misclassification cost by setting a high-cost value for the minority class and a small cost value for the majority class. The weight values of the cost matrix assigned to rare attacks are greater than the benign data that reflects their importance, which forces the classifier to identify these attacks in order to avoid high cost values due to misclassification errors. The cost weight can be the same value for more than one class as their samples are very small in the dataset, as explained in Table 4.

### 6.2. AdaBoost iterations

To determine the optimal number of AdaBoost iterations required to build the final AdaBoost classifier, the proposed algorithm executes the
calculate the optimal depth of the decision tree, the algorithm executes many tests with different decision tree depths ranging from 2 to 20 decision trees. The algorithm computes the performance metrics for each depth.

Decision tree depth versus macro F1.

Fig. 15 depicts the macro F1-score for various values of the decision tree depth using the two datasets. According to this graph, as the decision tree depth increases, the more accurate the macro F1-score until the depth of 10 layers, then the enhancement on the F1-score is very tiny. So, based on these results, the algorithm will use a decision tree with a depth of 10 that achieve a better macro F1-score while minimizing the computational cost and the execution time.

Table 6 summarizes the macro F1-score and G-mean performance metrics versus various decision tree depths. The results prove that a decision tree depth of 10 layers achieves optimized performance while reducing the computational cost and execution time. When the decision tree depth exceeds 10 layers, the algorithm slightly improves the macro F1-score and G-mean performance, but at a high computational cost and execution time.

6.4. Attack classification

This part presents a comparison of the performance of the proposed algorithm against four previous methods that focus on handling class imbalance: adaptive boosting (AdaBoost) (Xiao and Xiao, 2019), boosting with random undersampling (RUSBoost) (Xiao and Xiao, 2019), boosting with oversampling using SMOTE (SOMTEBoost) (Karatatas et al., 2020), and random forest (Gao et al., 2019). The metrics utilized to evaluate the performance are macro F1-score and G-mean using CSE–CIC–IDS2017, as in Table 7, and CSE–CIC–IDS2018, as in Table 8.

The results in the previous two tables indicate that the proposed algorithm outperforms other methods as it can handle the class imbalance and detect rare attacks. The proposed algorithm achieves the macro F1-score of 0.99 and the G-mean of 0.992 using CIC-IDS2017, while for CIC-IDS2018 the macro F1-score is 0.993 and the G-mean is 0.994. Therefore, macro F1 and G-mean improved as the proposed algorithm focuses on the misclassified instances in the minority class, which can prevent the minority class attacks from being wrongly classified as noise in the majority class and classified as normal traffic. The performance of random undersampling is low as it deletes valuable samples, while oversampling achieves good results, although it needs high computing power. The random forest technique does not address the class imbalance, and therefore its performance is low compared with other methods.

Table 9 and Table 10 present a comparison of the proposed algorithm against the previous methods per attack class using the macro F1-score metric.
Table 9 represents the results of the CSE–CIC–IDS2017 dataset, while Table 10 represents the results of the CSE–CIC–IDS2018 dataset. Tables 9 and 10 show the results of the proposed algorithm against the previous methods per attack class using the macro F1-score metric. For the benign traffic classification, the RF forest is the best algorithm, and the proposed method is the second, with a difference of 0.003. However, classifying normal traffic as an attack is less dangerous than classifying an attack as normal traffic, so the difference is very small and not a severe issue. For all attack types, the proposed algorithm outperforms all other algorithms, while the RF performance is degraded as it cannot handle the imbalance problem of the dataset. The proposed method sets a higher weight value for the samples of rare attack samples than normal traffic samples, which increases the misclassification cost of minority class samples. As a result, the macro-F1 value of the minority class samples is enhanced.

Summarizing the results of the table and sorting the various algorithm performances for classifying rare attacks, the proposed method is the best, followed by the SMOTE boost, the AdaBoost, the RUS boost, and finally the RF, which is the worst classifier for rare attacks as it ignores handling the class imbalance problem while the four algorithms handle the class imbalance issue.

Tables 11 and 12 represent the results of detecting individual classes and determining the attack type using the G-mean. Table 12 represents the results of the CSE–CIC–IDS2017 dataset, while Table 12 represents the results of the CSE–CIC–IDS2018 dataset.

As shown in Tables 11 and 12, the algorithm evaluates the performance using G-mean per each class type using the proposed method against the previous methods for both the datasets. The G-mean metric is used to evaluate the algorithm’s performance on multiclass imbalanced classification problems because the G-mean metric combines both sensitivity
and specificity by taking their geometric mean. The results indicate that the proposed method enhances the detection of all attack types as it can detect rare attacks without reducing familiar attacks or normal traffic. The random forest cannot handle the rare attacks compared with other methods. In most experiment results, the proposed algorithm outperforms other methods and achieves the highest performance to detect rare attacks, followed by AdaBoost, then oversampling using SMOTE with boosting techniques, next random undersampling with boosting techniques, and finally, random forest. This is because the proposed algorithm assigns a large weight to the rare intrusion samples, which is equivalent to increasing the misclassification cost of minority class samples. Therefore, the G-mean of the minority class samples is improved. From these two tables, it can be seen that the proposed algorithm performs the best among the other algorithms in general.

The detection results of the rare attacks using the CSE–CIC–IDS2018 dataset are better than the results obtained using the CSE–CIC–IDS2017 dataset as the size of CSE–CIC–IDS2018 is larger than CSE–CIC–IDS2017, about 5.7 times larger, which enables the classifier to learn enough from the samples. This big size has one drawback as it needs a lot of processing power to reduce the classification time. This can meet the objective of this paper of detecting all types of attacks, both rare and familiar, with high accuracy.

The computational complexity of the algorithms as the main goal of successful ML classification algorithm is to simultaneously maximize the classification metrics while minimizing the runtime resources consumption. However, these are often

<p>| Table 11. G-mean comparison per attack class in CSE–CIC–IDS2017 dataset. |
|----------------------------------|----------------|----------------|----------------|----------------|----------------|</p>
<table>
<thead>
<tr>
<th>Class label</th>
<th>Proposed method</th>
<th>AdaBoost</th>
<th>RUSBoost</th>
<th>SOMTE boost</th>
<th>RF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benign</td>
<td>0.994</td>
<td>0.985</td>
<td>0.974</td>
<td>0.994</td>
<td>0.997</td>
</tr>
<tr>
<td>DoS Hulk</td>
<td>0.988</td>
<td>0.983</td>
<td>0.97</td>
<td>0.99</td>
<td>0.84</td>
</tr>
<tr>
<td>PortScan</td>
<td>0.988</td>
<td>0.982</td>
<td>0.968</td>
<td>0.99</td>
<td>0.75</td>
</tr>
<tr>
<td>DDoS</td>
<td>0.988</td>
<td>0.983</td>
<td>0.967</td>
<td>0.99</td>
<td>0.75</td>
</tr>
<tr>
<td>DoS GoldenEye</td>
<td>0.991</td>
<td>0.984</td>
<td>0.968</td>
<td>0.99</td>
<td>0.75</td>
</tr>
<tr>
<td>FTP-Patator</td>
<td>0.991</td>
<td>0.982</td>
<td>0.965</td>
<td>0.99</td>
<td>0.6</td>
</tr>
<tr>
<td>SSH-Patator</td>
<td>0.991</td>
<td>0.98</td>
<td>0.965</td>
<td>0.99</td>
<td>0.62</td>
</tr>
<tr>
<td>DoS slowloris</td>
<td>0.991</td>
<td>0.98</td>
<td>0.964</td>
<td>0.99</td>
<td>0.61</td>
</tr>
<tr>
<td>DoS Slowhttp</td>
<td>0.991</td>
<td>0.98</td>
<td>0.963</td>
<td>0.99</td>
<td>0.55</td>
</tr>
<tr>
<td>Bot</td>
<td>0.993</td>
<td>0.98</td>
<td>0.964</td>
<td>0.99</td>
<td>0.5</td>
</tr>
<tr>
<td>Web Attack-Brute Force</td>
<td>0.993</td>
<td>0.978</td>
<td>0.96</td>
<td>0.99</td>
<td>0.42</td>
</tr>
<tr>
<td>Web Attack-XSS</td>
<td>0.993</td>
<td>0.978</td>
<td>0.96</td>
<td>0.99</td>
<td>0.4</td>
</tr>
<tr>
<td>Infiltration</td>
<td>0.993</td>
<td>0.975</td>
<td>0.955</td>
<td>0.99</td>
<td>0.4</td>
</tr>
<tr>
<td>Web Attack-SQL Injection</td>
<td>0.993</td>
<td>0.975</td>
<td>0.95</td>
<td>0.99</td>
<td>0.35</td>
</tr>
<tr>
<td>Heartbleed</td>
<td>0.993</td>
<td>0.973</td>
<td>0.95</td>
<td>0.99</td>
<td>0.34</td>
</tr>
</tbody>
</table>

<p>| Table 12. G-mean comparison per attack class in CSE–CIC–IDS2018 dataset. |
|----------------------------------|----------------|----------------|----------------|----------------|----------------|</p>
<table>
<thead>
<tr>
<th>Class label</th>
<th>Proposed method</th>
<th>AdaBoost</th>
<th>RUSBoost</th>
<th>SOMTEBoost</th>
<th>RF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benign</td>
<td>0.994</td>
<td>0.985</td>
<td>0.974</td>
<td>0.994</td>
<td>0.997</td>
</tr>
<tr>
<td>DoS Hulk</td>
<td>0.988</td>
<td>0.983</td>
<td>0.97</td>
<td>0.99</td>
<td>0.84</td>
</tr>
<tr>
<td>PortScan</td>
<td>0.988</td>
<td>0.982</td>
<td>0.968</td>
<td>0.99</td>
<td>0.75</td>
</tr>
<tr>
<td>DDoS</td>
<td>0.988</td>
<td>0.983</td>
<td>0.967</td>
<td>0.99</td>
<td>0.75</td>
</tr>
<tr>
<td>DoS GoldenEye</td>
<td>0.991</td>
<td>0.984</td>
<td>0.968</td>
<td>0.99</td>
<td>0.75</td>
</tr>
<tr>
<td>FTP-Patator</td>
<td>0.991</td>
<td>0.982</td>
<td>0.965</td>
<td>0.99</td>
<td>0.6</td>
</tr>
<tr>
<td>SSH-Patator</td>
<td>0.991</td>
<td>0.98</td>
<td>0.965</td>
<td>0.99</td>
<td>0.62</td>
</tr>
<tr>
<td>DoS slowloris</td>
<td>0.991</td>
<td>0.98</td>
<td>0.964</td>
<td>0.99</td>
<td>0.61</td>
</tr>
<tr>
<td>DoS Slowhttp</td>
<td>0.991</td>
<td>0.98</td>
<td>0.963</td>
<td>0.99</td>
<td>0.55</td>
</tr>
<tr>
<td>Bot</td>
<td>0.993</td>
<td>0.98</td>
<td>0.964</td>
<td>0.99</td>
<td>0.5</td>
</tr>
<tr>
<td>Web Attack-Brute Force</td>
<td>0.993</td>
<td>0.978</td>
<td>0.96</td>
<td>0.99</td>
<td>0.42</td>
</tr>
<tr>
<td>Web Attack-XSS</td>
<td>0.993</td>
<td>0.978</td>
<td>0.96</td>
<td>0.99</td>
<td>0.4</td>
</tr>
<tr>
<td>Infiltration</td>
<td>0.993</td>
<td>0.975</td>
<td>0.955</td>
<td>0.99</td>
<td>0.4</td>
</tr>
<tr>
<td>Web Attack-SQL Injection</td>
<td>0.993</td>
<td>0.975</td>
<td>0.95</td>
<td>0.99</td>
<td>0.35</td>
</tr>
<tr>
<td>Heartbleed</td>
<td>0.993</td>
<td>0.973</td>
<td>0.95</td>
<td>0.99</td>
<td>0.34</td>
</tr>
</tbody>
</table>
conflicting objectives and highly accurate algorithms often requiring long runtimes, which is not acceptable for IDS real-time security solutions. So, it must consider how much resources are needed to run the algorithm when evaluating a classifier for data classification. Table 13 shows the runtime consumption of the training and testing time for the proposed algorithm against other methods when handling the used dataset to detect attacks.

There are many observations from this table, with regard to training time; all models took longer to train with CSE–CIC–IDS2018 as the data source than CIC-IDS2017 due to the large record volume of the dataset. SMOTE-based approaches require the highest computational resources, as oversampling increases the size of the dataset by generating a large number of artificial instances. In addition, the increase in computational cost lies in the oversampling method itself, as the SMOTE-based approaches rely on the nearest neighbor computation to generate artificial instances, which leads to significant increases in their complexity.

Undersampling-based approaches do not consider all available training data, which corresponds to a loss of available information. Moreover, it is not guaranteed that the subset of data includes a sufficient number of critical samples close in the regions where classes overlap. So, in undersampling, the computational load is decreased, and both training and testing become very small compared with other techniques.

The reason for the large training time of random forest is that in random forest, a certain number of full-sized trees are grown on different subsets of the training dataset, while AdaBoost uses stumps. Each tree in the random forest is made up of all the features in the dataset, while stumps use one feature at a time. The primary concept behind RF is that it combines a single network of the companions of multidecision trees into a single model. With a large dataset, however, the training time will be very long as it generates many trees that require high computational power and resources.

7. Conclusion

Intrusion detection is considered a major challenge for research because the number of attack packets is small compared with benign data packets, which generate an imbalanced multiclass dataset. This paper proposed an enhanced rare attack detection algorithm in the cloud data center based on SDN architecture using AdaBoost ensemble learning and cost-sensitive techniques. AdaBoost is used in various classification problems and may be integrated with other techniques to enhance classification performance. However, because its methodology focuses on overall classification accuracy rather than the accuracy of the classification per individual class in the dataset, its classification accuracy is limited when handling imbalanced data and can easily cause overfitting.

This paper proposes AdaBoost and a cost-sensitive algorithm to detect rare attacks using weight vectors designed to optimize the original AdaBoost algorithm. It precisely inherits the advantages of AdaBoost by merging the results of 100 decision tree classifiers acting as the base classifiers. The cost-sensitive plays a crucial role in the AdaBoost iteration learning process to force the classifier to detect rare attacks; otherwise, the cost value will be expensive and harmful. The genetic algorithm is used to optimize the cost-matrix vector. The proposed algorithm considers both per-class and overall detection performance. The proposed algorithm outperforms the existing methods in terms of recall, precision, and F-1 measures. In addition, it minimizes overfitting and is very stable compared with other methods.

The potential future work of this paper may include the following. Measure and compare the computational cost of the proposed algorithm against other methods. The decision tree algorithm was used in this paper as a base classifier in AdaBoost, whereas any other ML algorithm can be used as a base classifier. Therefore, the performance of the boosting techniques with different base classifiers can be targeted in future studies. Although this
paper uses many evaluation metrics; however, various metrics can be used as the Kappa metric. Finally, extend the proposed technique of handling multiclass imbalanced datasets to other domains such as internet of things (IoT) security, rare diseases diagnosis in medicine, and product quality in the industry.

Conflict of interest

There are no conflicts of interest.

Credit authorship contribution statement

Authors contribution: Ahmed M. El-Shamy, Nawal A. El-Fishawy, Gamal M. Attiya, and Mokhtar Ahmed did the following: 1. Research idea proposal. 2. Research methodology. 3. Survey previous research. 4. Data collection and analysis. 5. Research idea development. 6. Simulation testing. 7. Final approval of the version to be published. Ahmed M. El-Shamy and Mokhtar Ahmed did the following: 1. Research idea development. 2. Drafting the article. 3. Investigation and writing the research paper.

References


