Computers, Materials & Continua
DOI:10.32604/cmc.2022.019667
Article

Tech Science Press

Real-Time Network Intrusion Prevention System Using Incremental Feature Generation

Yeongje Uhm[1] and Wooguil Pak[2,*]

[1]Research and Business Development Foundation, Yeungnam University, Gyeongsan, Gyeongbuk, 38541, Korea
[2]Department of Information and Communication Engineering, Yeungnam University, Gyeongsan, Gyeongbuk, 38541, Korea
[*]Corresponding Author: Wooguil Pak. Email: wooguilpak@yu.ac.kr
Received: 21 April 2021; Accepted: 28 May 2021

Abstract: Security measures are urgently required to mitigate the recent rapid increase in network security attacks. Although methods employing machine learning have been researched and developed to detect various network attacks effectively, these are passive approaches that cannot protect the network from attacks, but detect them after the end of the session. Since such passive approaches cannot provide fundamental security solutions, we propose an active approach that can prevent further damage by detecting and blocking attacks in real time before the session ends. The proposed technology uses a two-level classifier structure: the first-stage classifier supports real-time classification, and the second-stage classifier supports accurate classification. Thus, the proposed approach can be used to determine whether an attack has occurred with high accuracy, even under heavy traffic. Through extensive evaluation, we confirm that our approach can provide a high detection rate in real time. Furthermore, because the proposed approach is fast, light, and easy to implement, it can be adopted in most existing network security equipment. Finally, we hope to mitigate the limitations of existing security systems, and expect to keep networks faster and safer from the increasing number of cyber-attacks.

Keywords: Network intrusion detection; network intrusion prevention; real-time; two-level classifier

1 Introduction

Recently, we have experienced a rapid increase in cybercrime [1]. With a growing number of people working from home, the importance of network security has increased. Therefore, there is an urgent need to develop new technology to keep networks and users safe from malicious attacks. Early network security technology uses signature-based detection to discover network attacks using specific patterns identified by analyzing previous attacks [2–4]. Signature-based detection can significantly increase detection speed, thereby enabling network intrusion prevention in real time, and intrusion detection in non-real time.

However, signature-based methods are considerably vulnerable to variants of existing attacks and to newly emerging attacks, especially zero-day attacks [5–7]. Therefore, machine learning-based technology for detecting abnormal behaviors (instead of a pattern-dependent method) has recently been developed to overcome these vulnerabilities. Although a number of studies are underway into the proposed machine learning-based technologies, most of them focus on improving detection accuracy; research on improving detection speed to achieve real-time detection is lacking [8]. This is evident from the fact that machine learning-based technologies are applied only to intrusion detection systems (IDSs). Thus far, intrusion prevention, which blocks intrusions in real time, does not have applicable systems using machine learning.

There might be several reasons why a machine learning-based intrusion prevention system (IPS) has not yet been developed; however, the most important reason is the complexity of the machine learning algorithm itself. Most machine learning algorithms are trained on large amounts of data, and classification is then performed by the generated models [9–14]. It requires a considerably long time to train machine-learning models with large amounts of data, and this requires huge amounts of memory and computing power. To solve these problems, various partitioning-based machine learning techniques have been proposed, and some of the problems can be solved by adopting external cloud systems to mitigate the lack of memory and computational power required by training procedures.

However, classification by a learning model requires considerable computing power and fast speeds. As a solution, high classification speed can be obtained by massively parallel processing using expensive multiple GPUs. In this case, the CPU-GPU latency from transferring and processing a large amount of data can be detrimental to a high-capacity network that needs to transmit packets at high speed without delay [15].

Furthermore, the biggest reason why a machine learning-based IPS is difficult to implement is that it takes too long to generate from network traffic the features used for machine learning. There are several approaches to generating features; however, most studies generate features from each session, rather than from single packets. In this case, features cannot be generated before the session ends, and any attack is detected after the session ends [5–7,16,17]. Thus, attacks cannot be detected in real time. Moreover, with the current approaches, it is even difficult to detect an attack soon after the session ends because of poor classification performance.

Thus, in this study, we propose a method of generating features and detecting attacks in real time before the session ends. In particular, this study makes the following contributions.

(1) A structure for generating features in real time

By presenting the structure for generating features in real time, the proposed method enables early attack detection by determining whether an attack has occurred before the session ends.

(2) High accuracy and real time attack detection through a two-level classifier

To detect an attack in real time, we propose a unique two-level detection method. We designed a first-stage classifier that can detect attacks at high speed; the second-stage classifier improves detection accuracy. Thus, we implement a classifier with high accuracy while detecting attacks in real time.

(3) Low implementation costs

Although the proposed method uses a two-level classifier, it has the advantage of being applicable to existing equipment because it uses the same classifier (or two similar classifiers) to simplify implementation, compared to other hybrid methods or ensemble-based classifier methods.

The remainder of this manuscript is organized as follows. Section 2 identifies and compares the features in existing work. Section 3 describes the proposed method in detail. Section 4 analyzes the results of the performance evaluation. Finally, Section 5 concludes this study with a brief summary.

2 Existing Work

Early network intrusion detection systems (NIDSs) use pattern-matching or threshold-based approaches. Such NIDSs can support fast detection but reveal crucial limitations in detecting zero-day attacks. Thus, a lot of research is focusing on machine learning-based approaches. The early machine learning-based NIDS employed a single machine learning algorithm, so it showed weakness in accurately detecting various network attacks. NIDS research using multiple machine learning algorithms has been actively going on. Generally, machine learning-based IDSs are classified into packet-based methods and session-based methods, where the former use packet data for learning, and the latter use session data. The packet-based methods obtain features from raw packet data without a feature extraction technique. The session-based methods require all the session data in order to build features after the session is finished or has expired. Since the information from one entire session is reduced to a small number of statistical values called session features, it can support very high processing speeds.

In this section, we describe in detail the existing work, from early non-machine learning-based approaches to various recent machine learning approaches. We also compare the pros and cons of each approach.

2.1 The Non-Machine-Learning Algorithm

The signature-based approaches can be classified into two groups according to whether they support real-time detection or not. One of the most well-known non-real-time detection methods using the signature-based approach is the earliest IDS for monitoring multi-user systems. It can detect some specific types of attack: intrusion attempts, unauthorized intrusions, data breaches, DDoS, and suspicious use. The security policy is converted into rules and stored in a database, and each flow is analyzed to determine whether it was an attack or not based on the data registered in the database. After the flow closes, features are extracted from transmitted and received packet data and are used for detection. Thus, this approach cannot support real-time detection [2].

One of the NIDSs belonging to this category can provide real-time intrusion detection and prevention using the Boyer–Moore pattern matching algorithm in a signature-based manner [2,18]. It compares the header, payload, and size of an incoming packet to pre-registered signatures to identify malicious traffic. However, the system has some issues, such as processing overhead and reliability. It needs to analyze every packet to create a new signature. Nonetheless, it cannot guarantee the reliability of a signature.

2.2 The Packet-Based Single-Machine-Learning Algorithm

This approach uses a single-machine-learning algorithm with features obtained from packet data [2]. From the packet-based features, it can detect malicious code in packet payload data similar to an early pattern-matching approach. However, it inherently cannot detect zero-day attacks and attack variants, and the NIDS using this approach can be bypassed via packet fragmentation to avoid detection. By collecting multiple packets of a session rather than a single packet, such a weakness can be mitigated.

2.3 The Packet-Based Multiple-Machine-Learning Algorithm

This approach adopts multiple machine-learning algorithms to detect attacks [3]. Multiple algorithms can greatly help increase classification performance but the classification speed can deteriorate. Thus, the main disadvantage of this approach is that it is very difficult to use in large networks because of the slow training and classification speeds [3].

2.4 The Session-Based Single-Machine-Learning Algorithm

This approach extracts features from each session and classifies each session to detect abnormal traffic [9–14]. Early machine learning-based studies belong to this category. Since it does not use packet data to generate features, but uses a fixed number of features (regardless of the session length or packet size of each session), it can reduce memory usage and simplify the classification algorithm, resulting in high training and classification speeds. Owing to such benefits, we can apply this approach to large-scale networks. However, features can only be generated after the session ends, so when it detects an attack, it has most likely already been completed.

2.5 The Session-Based Multiple-Machine-Learning Algorithm

This approach performs training and classification by using features extracted from a session by using various classification algorithms. Ensemble and multi-layered methods are well-known types in this category [17,19]. The ensemble method applies several algorithms and combines the results from them. By doing so, it can significantly improve the detection performance, compared to a single-machine-learning approach. The multi-layered method runs each algorithm serially, based on the results after executing a specific algorithm. Generally, this approach adopts unsupervised learning and supervised learning. One example applies k-nearest neighbors (kNN) at first, to obtain multiple partitions, and then applies a decision tree (DT) algorithm to each partition. A multiple-classification algorithm compensates for the weakness of each algorithm, reaching very high classification accuracy. Instead, the classification speed becomes too slow to support real-time attack detection because of the very high computational cost. For some algorithms in this category, it is even impossible to apply them to a real network security system, because the overall implementation cost is too high.

As of now, little research has been done to increase detection accuracy and speed simultaneously. Various approaches have been proposed for overcoming existing technical issues, but real-time detection is still an open problem.

3 Proposed Algorithm

We propose a method for implementing an NIDS that can process packets received in real time and determine whether an attack has occurred. The proposed algorithm generates the latest features by updating the feature table for each session whenever a packet is received, and it determines whether an attack has occurred using the features. As shown in Fig. 1, the proposed system is configured to simultaneously increase both classification speed and accuracy by utilizing two classifiers. The proposed method has the following features.

- **Early attack detection**

The proposed method performs intrusion detection whenever a packet is received. Therefore, it can detect intrusions without waiting until the session ends.

- **Easy implementation**

Although the proposed method is equipped with two classifiers, it is implemented using the simple DT and its variants. Hence, it is considerably easy and simple to implement, and therefore, it is possible to apply the proposed method, without high cost, to an existing system.

The proposed method consists of a classifier to apply whenever a packet is received, and another classifier to apply when a session has ended. The classification executed whenever a packet is received is done by the cumulative packet-based classifier (CPC), and the classification executed after the session ends is done by the terminated flow-based classifier (TFC). A session is composed of a series of two-way packets. Therefore, session f is denoted as $f = \{p_1, p_2, \ldots, p_n\}$ based on the sequence of two-way packets received by the IDS. Here, f is a session consisting of n packets. The session is defined based on a five-tuple, <sip, dip, sport, dport, protocol>, in which sip, dip, sport, and dport denote the source IP, destination IP, source port, and destination port, respectively. Thus, $<ip_1$, ip_2, $port_1$, $port_2$, $protocol>$ and $<ip_2$, ip_1, $port_2$, $port_1$, $protocol>$ are regarded as the same session if the lifetimes overlap.

Whenever an IDS or IPS receives a packet, it creates and updates session statistics to generate features for the relevant session. Now, suppose F_k is the feature vector generated using the first k packets received. Assuming the total number of packets of the session is n, a total of n pairs of feature vectors are created for the session (i.e., $F_1, F_2, \ldots, F_n$). Here, the CPC uses $F_1, F_2, \ldots, F_n$, to classify whether the session is under attack, whereas the TFC uses only F_n to estimate abnormality. It is common to remove sip and dip from the features used to train the CPC and TFC. This is to prevent creation of a specific session-dependent model. Furthermore, in the CPC, dport is excluded from the feature. Now, we describe in detail updating and generating features whenever a new packet is received. We also show how the CPC and TFC work.

3.1 Incremental Feature Generation

Whenever a packet is received, the proposed algorithm updates information on the session to which the packet belongs, and creates the features required for classification. As shown in Fig. 1, the session information is stored in the feature table, which consists of internal session states and session stateful features. Internal session states are not features, but rather, the information necessary to create features. For example, Last Flow Timestamp (a field included in internal session states) stores the time at which every packet is received. This value is then used to update other values of internal session states or to create other features.

The internal session state is composed of bi-directional flow information and uni-directional flow information, i.e., forward and backward flow information. Whenever a packet is received, the corresponding fields for bi-directional flow information are always updated. Subsequently, fields for forward or backward flow information are updated according to the direction of the packet. Tab. 1 shows some selected fields for bi-directional flow information and shows how they are updated whenever a packet is received. Similarly, Tab. 2 shows a partial set of the fields for forward information, and how to update them. We omit fields for backward flow information since they are almost identical to the forward ones.

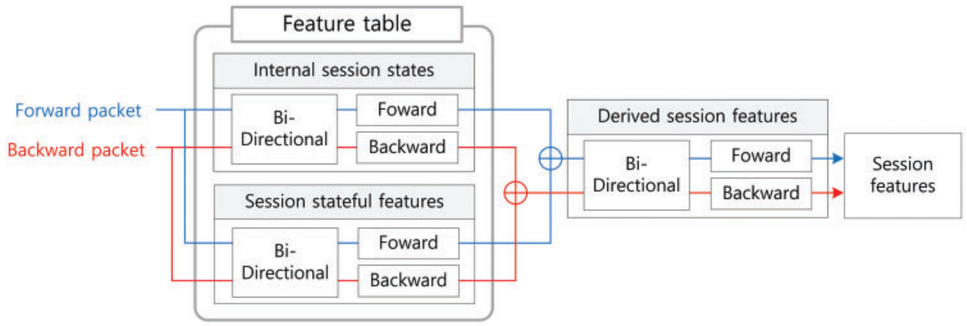


Figure 1: Overall procedures in incremental feature generation

Table 1: Partial list of bi-directional internal session states

No.	Field name	Calculation	Explanation
1	Last Flow Timestamp	= current packet timestamp	Timestamp of the last received packet
2	Flow IAT sqr	= v + Flow IAT^2 where Flow IAT' = Last Flow Timestamp*− Last Flow Timestamp#	Total sum of IAT^2
3	Pkt Len sqr	= v + current packet size^2	Total sum of packet size^2
4	Start Active Time	= Last Flow Timestamp* if Flow IAT* > 5 s	Start active time
5	Sub Flow Cnt	= v + 1 if Flow IAT* > 1 s	Total sub-flow count
...	...	...	...

#The asterisk (*) indicates a field value after the update. Therefore, Flow IAT and Flow IAT* are values before and after the update, respectively. In addition, 'v' denotes a value before the update of the described field.

Table 2: Partial list of forward internal session states

No.	Field name	Calculation	Explanation
1	Fwd Pkt Len sqr	= v + current packet size^2 if current packet is sent forward	Total sum of packet size^2 in the forward direction
2	Fwd Last Timestamp	= current packet timestamp if current packet is sent forward	Timestamp of the last received packet in the forward direction
3	Fwd IAT sqr	= v + Fwd IAT^{*2} where Fwd IAT* = Fwd Last Timestamp*− Fwd Last Timestamp	Total sum of IAT^2 in the forward direction
4	Fwd Last Blk TS	= current timestamp if no bulk timestamp exists	Last bulk timestamp in the forward direction
5	Fwd Blk Strt Hlpr	= current timestamp if no bulk timestamp exists	Current bulk timestamp internally used in the forward direction
...	...	...	...

We call information fields similar to the internal session state that work as features for classification the *session stateful features*. In session stateful features, bi-directional, forward flow, and backward flow information exists, and the corresponding fields are updated according to the direction of the received packets. Tab. 3 provides some selected fields of bi-directional flow information in session stateful features and shows how we update the fields.

Table 3: Partial list of bi-directional session stateful features

No.	Field name	Calculation	Explanation
1	Flow Duration	= v + Flow IAT*	Flow duration
2	Flow IAT Mean	= (v · (Tot Pkts-1) + Flow IAT*)/Tot Pkts	Mean time between two packets in the flow
3	Flow IAT Max	= max(v, Flow IAT*)	Maximum time between two packets in the flow
4	Flow IAT Min	= min(v, Flow IAT*)	Minimum time between two packets in the flow
5	Pkt Len Min	= min(v, current packet size)	Minimum length of a packet
...	...	...	...

As mentioned earlier, internal session states and session stateful features are updated every time a packet is received. Here, we should note that session stateful features do not include all features required for machine learning and classification. It means that we need to create the remaining features using internal session states and session stateful features. Such features are called derived session features. They are not stored or maintained in the feature tables shown in Fig. 1, and are temporarily generated through internal session states and session stateful features whenever required. Derived session features contain fields for bi-directional, forward, and backward flows.

Tab. 4 shows some typical bi-directional derived session features, and they are created by using internal session states and session stateful features. This feature-generation approach allows the system to progressively build session features. Whenever a packet is received, internal session states and session stateful features are updated. When the entire feature set is needed, derived session features are easily created without a high cost. Generally, we incur high overhead to create the entire feature set after the session is terminated. However, incremental feature generation distributes the overhead over time.

Table 4: Partial list of bi-directional derived session features

No.	Field name	Calculation	Explanation
1	Flow Byts/s	= (TotLen Fwd Pkts + TotLen Bwd Pkts)/Flow Duration	Byte flow rate (number of bytes transferred per second)
2	Flow Pkts/s	= (Tot Fwd Pkts + Tot Bwd Pkts)/Flow Duration	Packet flow rate (number of packets transferred per second)
3	Flow IAT Std	= sqrt((Flow IAT sqr*/Tot Pkts) − Flow IAT Mean^{*2})	Standard deviation in the time between two packets in the flow
4	Down/Up Ratio	= Tot Fwd Pkts/Tot Bwd Pkts	Download and upload ratio

3.2 The Cumulative Packet-Based Classifier

If F_k ($k < n$) for a specific attack session partially reflects the characteristics of an attack, it is possible to detect the attack using F_k. Here, the smaller the value of k, the faster the attack can be classified; however, the probability of incorrect classification may also increase. The overall characteristics of a session can be identified more accurately with an increase in k, but more time is spent detecting attacks. Ultimately, it is necessary to decide when to perform classification for the session. The session is no longer processed if it is classified as an attack in the CPC, and the relevant packet and the subsequent packets received by the IDS are discarded. Therefore, it is necessary to be cautious when classifying an attack in the CPC. In general, when machine learning is employed to detect a network intrusion, the relationship between the sip and dip address values should be used to create a feature (for example, by determining if they are the same). However, sip and dip address values should be removed from the feature. To make the CPC more reliable in detecting attacks, all features that can affect the creation of a model dependent on the session itself should be removed. Hence, sip, dip, and dport are all removed in the proposed method, whereas only sip and dip are removed in the conventional methods.

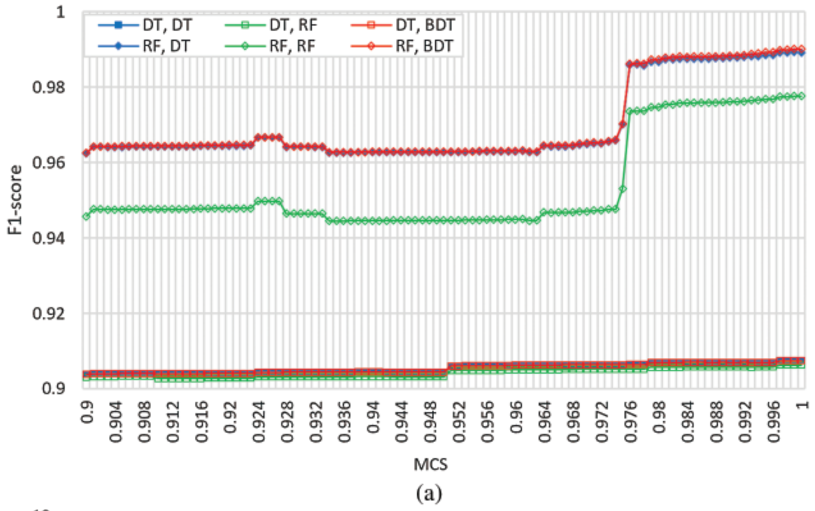
In general, class type and score are obtained as a result of CPC classification. The closer the score is to 1, the more reliable it is, whereas the closer the score is to 0, the more unreliable it is. Therefore, the minimum CPC score (MCS) should be determined—the higher the MCS, the lower the rate of misclassification by the CPC. However, with an increase in the number of packets used to generate features for classifying a session, it takes longer to detect an attack—the lower the MCS, the quicker the detection in the CPC. However, this leads to an increase in the probability of error. Therefore, in the proposed method, it is crucial to maintain high classification accuracy and to improve speed at the same time by setting the MCS to an optimal value.

3.3 The Terminated Flow-Based Classifier

The TFC and CPC use basically the same feature structure; however, unlike the CPC, the TFC performs classification after the session ends. Hence, there is no need to process the session in real time. Therefore, unlike the CPC, it is more advantageous for the TFC to use a classification algorithm with high accuracy rather than considering speed or computational complexity. Furthermore, while the CPC performs learning and classification using all of F_k ($k < n$), the TFC classifies only finished sessions. Therefore, in the TFC, learning and classification are performed using only the F_n features generated based on all packets of the finished session. This method uses the same features as those used in the CPC, but uses one more: dport.

3.4 Parameter Setting

As described above, the performance of the proposed method varies depending on the MCS. Therefore, after training, the optimal MCS value is set based on the results from classifying the training data. The proposed method uses decision tree algorithms for machine learning. In general, the decision tree algorithm is suitable for an IDS that processes large amounts of data owing to its fast training time, high classification speed, and low memory usage. Of the several decision tree algorithms, the most appropriate should be selected. Therefore, by considering three algorithms—DT, random forest (RF), and boosted DT (BDT)—we measure the F1-score while increasing the MCS value from a combination of each algorithm. Using these results, the optimal MCS for each algorithm was selected. The ISCXIDS2012 and CICIDS2017 datasets were used for the experiment. For reference, the measurement results using the CICIDS2017 dataset are shown in Fig. 2.



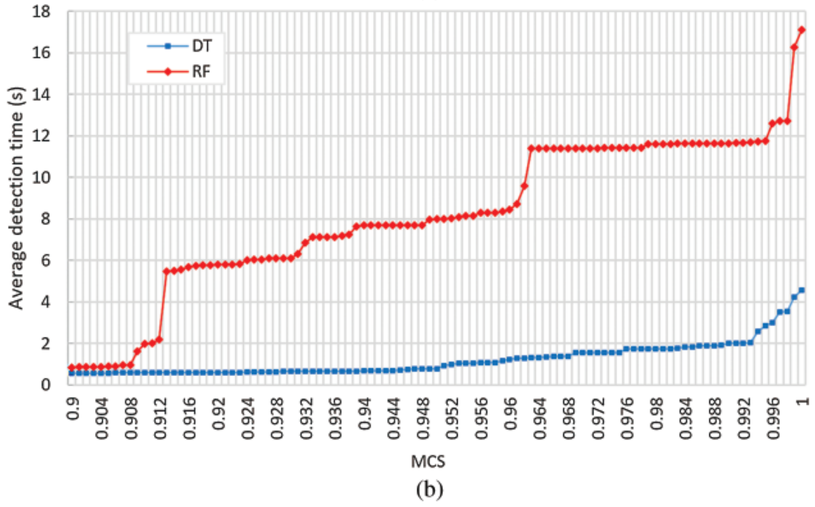


Figure 2: Results of the performance comparison based on the MCS from the CICIDS2017 dataset (a) F1-score based on the MCS and (b) Average detection time

As seen in Fig. 2, the F1-score was the highest when RF and BDT were used among the combinations of first- and second-level classifiers. Here, the F1-score was consistently maintained when the MCS was 0.977 or higher; however, the average detection time increased significantly, when the MCS was 0.998 or higher. Therefore, in the proposed method, we conducted experiments by setting the MCS to 0.998 when using RF for the CPC and BDT for the TPC, and by setting it to 1 when using DT and BDT. Similarly, the same method was used to select the best combination of classifiers and the relevant MCS for the ISCXIDS2012 dataset. Thus, the MCS was set to 0.985 when using RF and BDT, and it was set to 1 when using DT and BDT.

3.5 Overall System Operation

The overall operation of the proposed IPS is as follows. When a packet arrives, the IPS first determines whether to receive it or not according to the firewall policy. If the matched policy returns *deny*, it is discarded. Otherwise, it is accepted, and the internal session states and session stateful features are created or existing features are updated. After that, the system builds entire features for the packet after creating derived session features. It determines if the session for the packet is benign or not through the CPC. If the classification score is higher than MCS, the session is added to the firewall policy blacklist or whitelist based on the class type. Conversely, if it is lower than MCS, the packet is forwarded regardless of the classification result. When the session terminates, the internal session state and session stateful feature data for the session expire and are removed after building the final features. The final determination about the session is done by the TFC; the results are logged and the administrator is notified, if necessary. The overall operation is in Algorithm 1.

Algorithm 1: Intrusion Prevention System

```
1  IF packet P is received THEN
2      Consult firewall to find the matching policy for P.
3      IF policy action is 'deny' THEN
4          Drop P and RETURN
5      END_IF
6      Update bi-directional internal session states & bi-directional session stateful
7      features.
8      IF P is in the forward direction THEN
9          Update forward internal session states and forward session stateful features.
10     ELSE
11         Update backward internal session states and backward session stateful features.
12     END_IF
13         Create derived session features and classify P using CPC.
14     IF score < MCS THEN
15         RETURN
16     END_IF
17     IF P is malicious THEN
18         Add P session to blacklist of firewall.
19     ELSE
20         Add P session to whitelist of firewall.
21     END_IF
22 ELSE IF expired session S is found THEN
23     Create derived session features for S.
24     Remove data for S from the session table.
25     Classify S using TFC.
26 IF S is malicious THEN
27     Notify the administrator and log the result.
28 END_IF
```

4 Performance Evaluation

4.1 The Environment

To evaluate the performance of the proposed method, we compared its performance using various algorithms and two datasets: CICIDS2017 and ISCXIDS2012 [17,20]. For training and testing, the datasets were split in a 6:4 ratio. We chose these datasets because packet and labeling data are available, and therefore, features can be generated using CICFlowMeter. We used

80 features proposed in ISCXIDS2012. However, as described in Section 3, sip, dip, and dport were excluded from the first-level classifier, but only sip and dip were excluded from the second-level classifier. The size and characteristics of each dataset are summarized in Tab. 5.

Table 5: Characteristics for each dataset

ISCXIDS2012	Total pieces of data	163,806
	Total number of classes	5
	Ratio of the training and testing split	6:4
CICIDS2017	Total pieces of data	1,011,319
	Total number of classes	11
	Ratio of the training and testing split	6:4

For the performance comparison, we employed a 1D-CNN [21], LSTM [22], and TCN [23], which are deep learning algorithms [24], along with DT and Naïve Bayes (DTNB) as a clustering-based method [25], BDT as a boosted algorithm [26], and DT and RF [27], which are DT categories [18]. The parameter settings for each algorithm are listed in Tab. 6.

Table 6: Parameter settings for each algorithm

Algorithm	Parameter	Value
Proposed	MCS	ISCXIDS2012: 1 s for (DT, BDT), 0.985 s for (RF, BDT) CICIDS2017: 0.998
	Maximum depth	Unlimited
	Minimum samples to split	2
	Minimum samples leaf	1
	Maximum features	8
	Maximum leaf nodes	Unlimited
	Number of trees	RF: 10 BDT: 50
	AdaBoost	Learning rate: 1 Boosting algorithm: SAMME.R
1D-CNN	Learning rate	0.001
	Batch size	16
	Epochs	5
	Optimizer	Adam
	1-D convolution layer	Number of filters: 128; kernel size: 3; stride: 1
	Activation layer	ReLU
	Max pooling layer	Pooling size: 2
	1-D convolution layer	Number of filters: 128; kernel size: 1; stride: 1
	Activation layer	ReLU
	Max pooling layer	Pooling size: 2

(Continued)

Table 6: Continued

Algorithm	Parameter	Value
LSTM	Learning rate	0.001
	Batch size	32
	Epochs	5
	Optimizer	Adam
	Hidden layer	Number of neurons: 128; time step: 1
TCN	Learning rate	0.001
	Batch size	1024
	Epochs	100 (early stop enabled)
	Optimizer	Adam
	Number of filters	64
	Kernel size	2
	Number of residual block stacks	1
	Dilations	[1, 2, 4, 8, 16, 32, 64]
DTNB	Maximum depth	Unlimited
	Minimum samples to split	2
	Minimum samples leaf	1
	Maximum features	8
	Maximum leaf nodes	Unlimited

4.2 Comparison of Detection Rates

Of the various performance indicators in the classifiers used in the NIDS, the most crucial factor is detection rate. If normal and attack sessions cannot be accurately classified, such an algorithm is impractical for an NIDS, regardless of its high classification speed. In this experiment, we measured accuracy, precision, recall, and F1-score to compare the detection performance of each algorithm. The experimental results are shown in Figs. 3 and 4, which indicate that the results are similar, regardless of the dataset type. As seen in the figures, the proposed method using a combination of RF and BDT showed higher performance than the conventional competing methods for all metrics. Furthermore, the method using DT and BDT achieved slightly lower performance than the 1D-CNN and LSTM. In all cases, the proposed method using RF and BDT showed the highest accuracy and F1-score. This clearly demonstrates that the proposed two-level classifier structure is effective in improving accuracy.

4.3 Comparison of Detection Times

To detect an attack in real time, it should be possible to detect the attack before the session ends. To evaluate this capability, we measured and compared the time taken from the start of the session to detection of an attack. The shorter the time, the more effective the method is at detecting and defending against an attack in real time. Tab. 7 shows the results from comparing detection times for the proposed and comparison methods. Because all the competing methods are session-based IDSs, classification and detection were performed after the sessions ended.

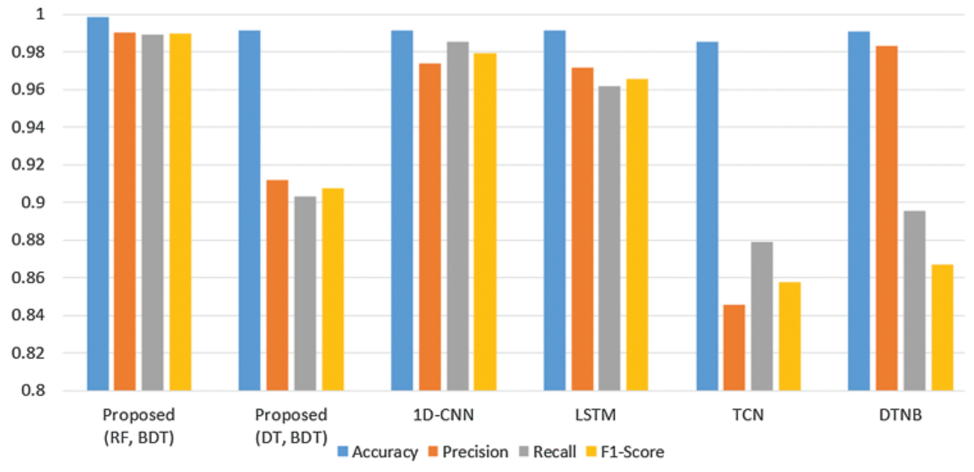


Figure 3: Experimental results using the CICIDS2017 dataset

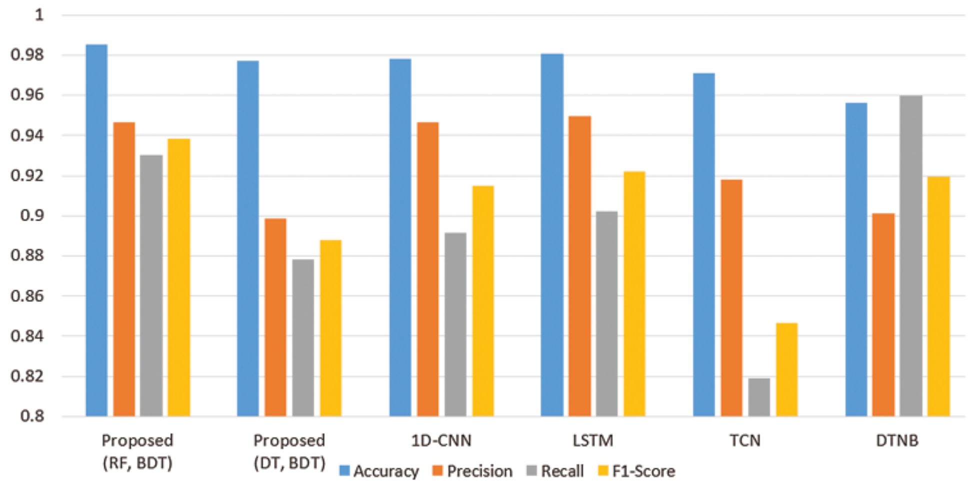


Figure 4: Experimental results using the ISCXIDS2012 dataset

Accordingly, in Tab. 7, the detection times of the session-based methods are expressed as the session duration, assuming no additional processing time existed. In an actual implementation, the comparison methods may take more time than the results shown in Tab. 7. The proposed method indicates the time taken for accurate detection after the session started.

As shown in Tab. 7, the proposed method makes use of the CPC to detect attacks even before the session ends. Hence, the speed at which an attack is detected by the CPC is a valid metric to gauge the performance of the comparison methods. Tab. 7 indicates that the proposed method can detect attacks significantly faster than the conventional methods. In particular, the (DT, BDT) method was at least five times faster than the (RF, BDT) method. The proposed method based on RF and BDT was almost three times faster than the conventional session-based methods. This clearly shows that the proposed method can provide detection speed that is not achievable with conventional methods. In particular, most session-based methods detect the session end using a timeout value. For the TCP, the session end time can be determined by detecting the FIN packet;

however, in the other protocols, such as a UDP session, the IDS cannot accurately detect the session end time. Therefore, it needs to estimate the end time after no packets are transmitted for predefined duration, which is generally set at 30 s to 120 s. For session-based IDSs in a real environment, the sum of the session duration and the timeout value is defined as the total detection time. Thus, the gap in the actual detection speed between the proposed method and a conventional session-based method becomes larger than that shown in Tab. 7.

Table 7: Average detection time for each class in the CICIDS2017 dataset (in seconds)

Class	Session-based IDS	Proposed using (RF, BDT)	Proposed using (DT, BDT)
Benign	64.2	7.7	0.9
Bot	38.5	38.2	2.7
DDoS	76.9	4.2	0.02
DoS GoldenEye	83.5	20.6	16.0
DoS Hulk	117.1	55.3	1.6
DoS Slowhttptest	109.8	66.8	4.1
DoS slowloris	116.0	39.1	21.9
FTP-Patator	64.5	0.2	0.04
PortScan	60.1	59.8	18.9
SSH-Patator	36.1	30.2	0.001
Web Attack + Brute Force	61.1	59.9	3.6
Cumulative Average	77.8	27.2	4.1

To compare detection speeds more accurately, it is necessary to compare the speed for each class. Tab. 7 also shows those detection speeds, and the proposed method detects each class much faster than the conventional methods. In particular, the proposed method using (DT, BDT) can significantly reduce the detection time, compared to the proposed method using (RF, BDT). As seen in the previous experiment, the performance from (DT, BDT) is slightly lower than from (RF, BDT) in terms of detection rate. Therefore, it is advantageous to use (RF, BDT) when detection rate is more important than speed. Conversely, it is better to use (DT, BDT) when speed is more critical than detection rate.

Tab. 8 shows the average detection time for each class in the ISCXIDS2012 dataset. As with CICIDS2017, the detection time can be significantly reduced compared to the conventional session-based methods, so it is more suitable to use (DT, BDT) instead of (RF, BDT) if high detection speed is needed.

The detection time is affected by the inter-packet time in a session. That is, with an increase in the inter-packet time in the same session, the detection time also increases. Therefore, instead of measuring the relative detection time, we can compare the performance more accurately by determining how many packets within each session are received before an attack is detected. Tab. 9 summarizes the average number of packets required to detect each class type in CICIDS2017.

Table 8: Average detection time for each class in the ISCXIDS2012 dataset (in seconds)

Class	Session-based IDS	Proposed using (RF, BDT)	Proposed using (DT, BDT)
Benign	64.1	6.4	3.0
BruteForceSSH	40.1	0.8	0.4
DDoS	59.8	23.5	7.7
HTTPDoS	71.6	30.5	8.5
Infiltration	62.6	46.3	14.2
Cumulative Average	61.7	20.4	4.0

Table 9: Average number of packets before detection of each class in CICIDS2017

Class	Session-based IDS	Proposed using (RF, BDT)	Proposed using (DT, BDT)
Benign	23.9	1.5	1.3
Bot	6.1	3.9	2.1
DDoS	7.7	4.4	2.3
DoS GoldenEye	9.5	4.5	3.1
DoS Hulk	9.5	3.9	1.5
DoS Slowhttptest	6.6	5.4	3.3
DoS slowloris	8.1	3.6	3.0
FTP-Patator	13.3	3.4	2.9
PortScan	2.0	2.0	2.0
SSH-Patator	27.2	3.0	2.9
Web Attack + Brute Force	16.8	6.1	3.6
Cumulative Average	14.7	2.6	1.6

Tab. 9 indicates that the proposed method requires considerably fewer packets to detect an attack than the conventional session-based methods. Moreover, we can see that the number of packets required for (RF, BDT) is not significantly different from (DT, BDT). Tab. 10 summarizes the results using ISCXIDS2012, which are similar to those for CICIDS2017.

Table 10: Average number of packets before detection of each class in ISCXIDS2012

Class	Session-based IDS	Proposed using (RF, BDT)	Proposed using (DT, BDT)
Benign	24.6	2.8	1.7
BruteForceSSH	10.1	3.6	1.0
DDoS	67.5	4.3	2.0
HTTPDoS	12.3	9.3	2.1
Infiltration	17.6	5.6	2.0
Cumulative Average	33.0	3.5	1.8

4.4 System Load

The proposed NIDS should repeatedly classify the session whenever a new packet is received until the class type of a specific session is detected. That is, unlike the conventional session-based NIDS that requires a one-time classification for each session, the proposed NIDS performs more classifications. This can result in significantly higher overhead in the system, compared to the conventional method. Thus, such increased overhead can be an obstacle to real-time processing. The number of packets needed to classify each session becomes the most crucial factor, and should be minimized.

For a more accurate analysis of system loads, the total number of packets included in a session, and the number of packets required before detection, are displayed for each session in Fig. 5. The figure shows that the average number of packets required for detection is less than five in most cases. In particular, for the sessions in which the total number of packets is very high (>1000), the number of packets required for detection tends to stay consistently small, without a significant increase.

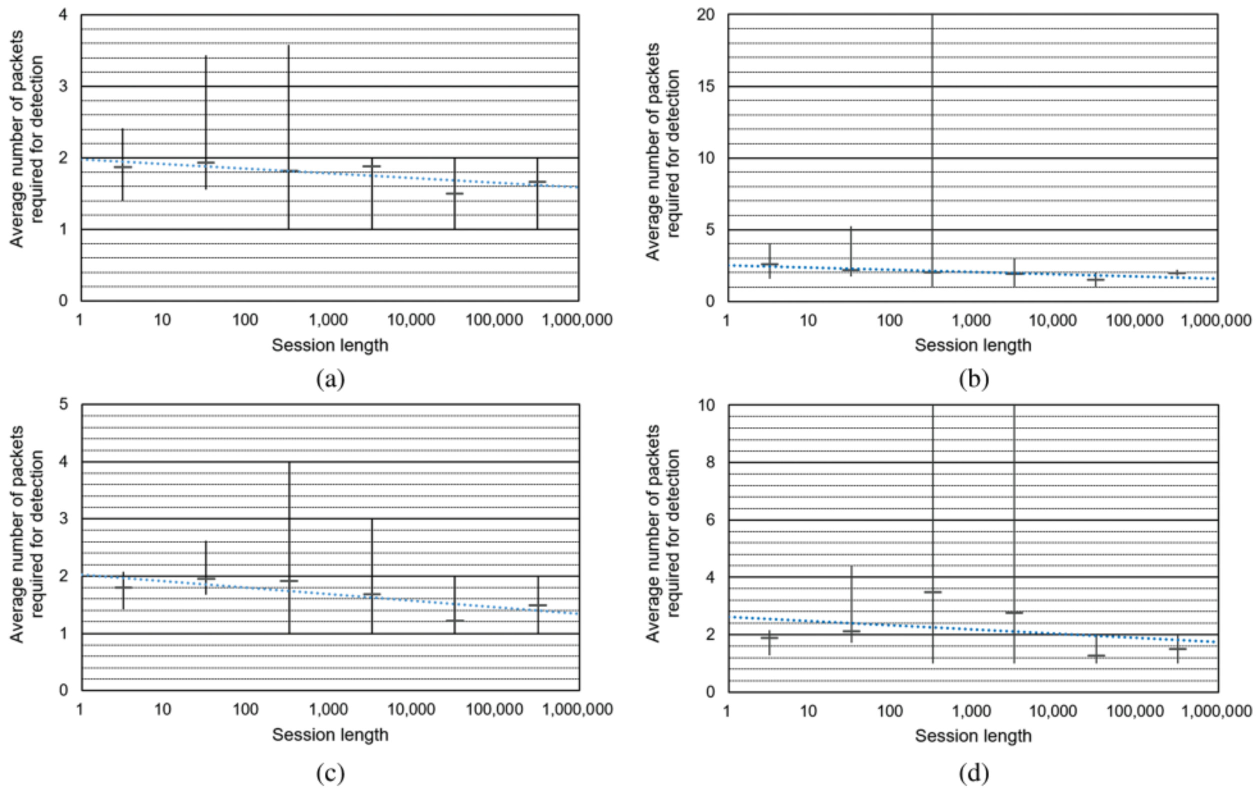


Figure 5: Length of each session and the number of packets required before detecting an attack. Vertical lines indicate the range of the number of packets. (a) CICIDS2017 (DT, BDT), (b) CICIDS2017 (RF, BDT), (c) ISCXIDS2012 (DT, BDT) and (d) ISCXIDS2012 (RF, BDT)

For example, when using (DT, BDT) for the CICIDS2017 dataset, we observed that, even for a session when the total number of packets was more than 100,000, it is possible to determine whether an attack has occurred with only the first two packets of the session. Thus, the proposed method can classify normal and attacked sessions while maintaining low system loads regardless of the session length. This is a significant characteristic for improving the performance of the NIDS. This characteristic demonstrates that real-time IPS development is a real possibility.

5 Conclusion

We proposed a new approach that can detect cyberattacks in real time. It is composed of two classifiers, one for processing packets in real time and the other for processing sessions in non-real time, so it can simultaneously increase detection performance in terms of speed and accuracy. In this research, we showed a promising solution enabling a machine learning-based real-time IPS rather than a machine learning-based non-real-time IDS by providing incomparable detection speed and accuracy. Of course, the proposed approach cannot process all the traffic and detect any kind of attack in real time. The hardware platform costs are higher than conventional IDSs since it requires almost twice the processing power, compared to the existing session-based approaches. However, despite these limitations, it is of great significance, showing that it is possible to implement real-time IPS-based rather than IDS-based machine learning algorithms. Future research will find solutions to the shortcomings revealed by this research. In doing so, we believe the proposed approach will improve so it is able to detect and defend against attacks in real time, even on 100-gigabit networks. We also expect that it can protect networks and users from malicious users and various network attacks.

Funding Statement: This work was supported in part by the Information Technology Research Center (ITRC) Support Program supervised by the Institute for Information and Communications Technology Planning and Evaluation (IITP) (IITP-2020-2016-0-00313), and in part by and the 2021 Yeungnam University Research Grant.

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

References

[1] D. Buil-Gil, F. Miró-Llinares, A. Moneva, S. Kemp and N. Díaz-Castaño, "Cybercrime and shifts in opportunities during COVID-19: A preliminary analysis in the UK," *European Societies*, vol. 23, no. Suppl. 1, pp. 1–13, 2021.

[2] M. Roesch, "Snort—Lightweight intrusion detection for networks," *Proceedings of the 13th USENIX Conference on System*, vol. 99, no. 1, pp. 229–238, 1999.

[3] W. Wang, Y. Sheng, J. Wang, X. Zeng, X. Ye *et al.,* "HAST-IDS: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection," *IEEE Access*, vol. 6, pp. 1792–1806, 2017.

[4] C. Kruegel and T. Toth, "Using decision trees to improve signature-based intrusion detection," in *Proc. the Int. Workshop on Recent Advances in Intrusion Detection*, Pittsburgh, PA, USA, pp. 173–191, 2003.

[5] S. X. Wu and W. Banzhaf, "The use of computational intelligence in intrusion detection systems: A review," *Applied Soft Computing*, vol. 10, no. 1, pp. 1–35, 2010.

[6] M. Ektefa, S. Memar, F. Sidi and L. S. Affendey, "Intrusion detection using data mining techniques," in *Proc. Information Retrieval & Knowledge Management (CAMP)*, Shah Alam, Selangor, Malaysia, pp. 200–203, 2010.

[7] L. Bilge and T. Dumitras, "Before we knew it: An empirical study of zero-day attacks in the real world," in *Proc. the 2012 ACM Conf. on Computer and Communications Security*, Raleigh North Carolina, USA, pp. 833–844, 2012.

[8] M. Al-Qatf, Y. Lasheng, M. Al-Habib and K. Al-Sabahi, “Deep learning approach combining sparse autoencoder with SVM for network intrusion detection,” *IEEE Access*, vol. 6, pp. 52843–52856, 2018.

[9] I. Ahmad, M. Basheri, M. J. Iqbal and A. Rahim, “Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection,” *IEEE Access*, vol. 6, pp. 33789–33795, 2018.

[10] S. Sahu and B. M. Mehtre, “Network intrusion detection system using J48 Decision Tree,” in *Proc. Int. Conf. on Advances in Computing, Communications and Informatics*, Kochi, India, pp. 2023–2026, 2015.

[11] Y. Cheong, K. Park, H. Kim, J. Kim and S. Hyun, “Machine learning based intrusion detection systems for class imbalanced datasets,” *Journal of the Korea Institute of Information Security & Cryptology*, vol. 27, no. 6, pp. 1385–1395, 2017.

[12] C. Yin, Y. Zhu, J. Fei and X. He, “A deep learning approach for intrusion detection using recurrent neural networks,” *IEEE Access*, vol. 5, pp. 21954–21961, 2017.

[13] K. Park, Y. Song and Y. Cheong, “Classification of attack types for intrusion detection systems using a machine learning algorithm,” in *Proc. 2018 IEEE Fourth Int. Conf. on Big Data Computing Service and Applications (BigDataService)*, Bamberg, Germany, pp. 282–286, 2018.

[14] W. Lin, H. Lin, P. Wang, B. Wu and J. Tsai, “Using convolutional neural networks to network intrusion detection for cyber threats,” in *Proc. 2018 IEEE Int. Conf. on Applied System Invention*, Taiwan, China, pp. 1107–1110, 2018.

[15] S. Han, K. Jang, K. Park and S. Moon, “PacketShader: A GPU-accelerated software router,” *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 195–206, 2010.

[16] L. Li, Y. Yu, S. Bai, Y. Hou and X. Chen, “An effective two-step intrusion detection approach based on binary classification and k-NN,” *IEEE Access*, vol. 6, pp. 12060–12073, 2017.

[17] S. Soheily-Khah, P. P. Marteau and N. Béchet, “Intrusion detection in network systems through hybrid supervised and unsupervised machine learning process: A case study on the ISCX Dataset,” in *Proc. the 1st Int. Conf. on Data Intelligence and Security*, South Padre Island (SPI), Texas, USA, pp. 219–226, 2018.

[18] R. Sedgewick, *Algorithms in C: Parts 1–4, Fundamentals, Data Structures, Sorting, and Searching*, 3rd ed., Boston, USA: Addison-Wesley Longman Publishing, pp. 702, 1997.

[19] Y. Yuan, L. Huo and D. Hogrefe, “Two layers multi-class detection method for network intrusion detection system,” in *Proc. IEEE Symp. on Computers and Communications*, Heraklion, Greece, pp. 767–772, 2017.

[20] I. Sharafaldin, A. H. Lashkari and A. A. Ghorbani, “Toward generating a new intrusion detection dataset and intrusion traffic characterization,” in *Proc. 4th Int. Conf. on Information Systems Security and Privacy*, Portugal, 2018.

[21] M. Azizjon, A. Jumabek and W. Kim, “1D CNN based network intrusion detection with normalization on imbalanced data,” in *Proc. Int. Conf. on Artificial Intelligence in Information and Communication*, Fukuoka, Japan, pp. 218–224, 2020.

[22] S. A. Althubiti, E. M. Jones and K. Roy, “LSTM for anomaly-based network intrusion detection,” in *Proc. 28th Int. Telecommunication Networks and Applications Conference*, Sydney, NSW, pp. 1–3, 2018.

[23] L. Zhipeng, Z. Qin, P. Shen and L. Jiang, “Intrusion detection using temporal convolutional networks,” in *Proc. Int. Conf. on Neural Information Processing*, Sydney, NSW, Australia, 2019.

[24] J. Cheng, Y. Liu, X. Tang, V. S. Sheng, M. Li *et al.,* “DDoS attack detection via multi-scale convolutional neural network,” *Tech Science Press Computers, Materials & Continua*, vol. 62, no. 3, pp. 1317–1333, 2020.

[25] M. Hall and E. Frank, “Combining naive Bayes and decision tables,” in *Proc. the 21st Int. Florida Artificial Intelligence Research Society Conf., FLAIRS-21*, Coconut Grove Florida, USA, 2008.

[26] Y. Coadou, “Boosted decision trees and applications,” *EPJ Web of Conferences*, vol. 55, no. 3, pp. 2004, 2013.

[27] V. Svetnik, A. Liaw, C. Tong, J. C. Culberson, R. P. Sheridan *et al.,* “Random forest: A classification and regression tool for compound classification and QSAR modeling,” *Journal of Chemical Information and Computer Sciences*, vol. 43, no. 6, pp. 1947–1958, 2003.