An Ensemble Machine Learning Botnet Detection Framework Based on Noise Filtering

Tzong-Jye Liu, Tze-Shiun Lin, Ching-Wen Chen

Department of Information Engineering and Computer Science, Feng Chia University, Taiwan
tjliu@fcu.edu.tw, zxkoaqes@gmail.com, chingwen@fcu.edu.tw

Abstract

During the past decade, one of the most serious cyber threats has been the growth of botnet. Since botnet attacks combine the characteristics of many malicious attacks, they have complex attack behaviors and communication patterns. In order to improve the detection rate, many researchers use machine learning techniques. In this paper, we proposed an ensemble classification framework based on noise filtering to improve detection performance. The experimental results show that the proposed framework improves the detection rate and reduces the false alarm rate. We also compare the proposed classification model with other ensemble classification models. The experimental results also show that the classification model has the highest accuracy and lower false alarm rate.

Keywords: Botnet detection, Ensemble classifier, Intrusion detection, Machine learning, Noise filtering

1 Introduction

With the rapid development of applications on the Internet, more and more cyber threats exist. These cyber threats include spam [1], Trojan horses [2], distributed denial of service [3], botnet [4], etc. One of the most serious cyber threats is the growth of botnets. A botnet is a collection of hosts that are infected by the malware. Infections on these hosts are usually carried out through known vulnerabilities and the infected hosts are called bots. Bots are usually controlled by the botmaster to perform the cyberattack.

Although the botnet detection technology [5-6] has been developed for many years, there is still no effective solution to eliminate botnets. After hackers modify attack parameters or malicious programs, the existing detection solutions will fail. Recently, researchers have begun to detect cyberattacks based on machine learning as well as deep learning techniques [7-11]. These detection solutions can identify existing attacks or even unknown attacks.

When we want to identify cyberattacks by using the machine learning techniques, we have to train the classification model by using the training dataset first. The training dataset is a set of feature vectors extracted from botnet and normal network behaviors. After establishing the classification model, we can identify cyberattacks by using the classification model. It is a challenge to design a classification model for detecting cyberattacks because the real network behavior is highly variable.

In order to prevent detection by classification models, botnet flows usually pretended to be normal flows. Therefore, many cyberattacks have the same characteristics as normal flow behavior. The normal and botnet flows that have similar characteristics are the *noise* flows. After the features of these flows are extracted and put into a dataset, noise [12-13] may be formed in the dataset. These noises will reduce the *detection rate* of the classification model and also increase its *false alarm rate*. The *detection rate* is the ratio of the number of correctly detected botnet flows to the total number of botnet flows. The *false alarm rate* is the ratio of the number of misclassified normal flows to the total number of normal flows.

When we want to identify the cyberattack on the Internet by using a classification model, designing a classification model with a high detection rate is the main goal. However, if the false alarm rate of the classification model is too high, system administrators will lose trust in the classification model due to a large number of false alarms. Therefore, the second goal of designing a classification model is to reduce the false alarm rate.

The motivation of this paper is to design a classification model that considers both low false alarm rate and high detection rate. To achieve this goal, we will start by filtering the noise in the input flows. For a given classification algorithm Ω, the proposed classification framework is an ensemble classification model. It first classifies the input flows into classes T and F. Class T contains flows that are correctly classified by the algorithm Ω, and class F contains noise flows that will be misclassified by the algorithm Ω. Therefore, in order to improve the overall

*Corresponding Author: Tzong-Jye Liu; E-mail: tjliu@fcu.edu.tw
DOI: 10.53106/160792642021112206012

performance, flows in class *F* will be classified using another classification algorithm, which is trained by using the classification result of flows in class *T*.

The contributions of this study are as follows: (1) This paper proposes an ensemble classification framework based on noise filtering to improve classification performance. Experimental results show that the framework achieves both low false alarm rate and high detection rate. (2) We compare the proposed classification model with other ensemble classification models. The experimental results also show that the classification model has the highest accuracy rate and lower false alarm rate. (3) Even after testing different types of botnet flows, the false alarm rate is still very small. (4) The *t*-distributed stochastic neighbor embedding (*t*-SNE) visualization technique is used to understand the characteristics of the dataset.

The following of this paper is organized as follows. Section 2 discusses related work. The proposed framework is in Section 3. The experimental results of the proposed framework are in Section 4. Finally, we give conclusion and future work in Section 5.

2 Related Work

The network intrusion detection system was first proposed in 1980 [14]. It can identify malicious behaviors and block them to protect users on the Internet. Recently, botnet detection [4] has become the main research direction of network intrusion detection. Since attacks of botnet combine the characteristics of many malicious attacks, they have complex attack behaviors and communication patterns [15]. In order to improve the detection rate, many researchers use machine learning and deep learning techniques [7-11, 16]. In [17], the co-training method is used for network intrusion detection. A small number of known training samples are used to predict unlabeled samples, and these samples are added to the training dataset for retraining in an iterative manner. The procedure improves the accuracy of the detection system when there is a limited training sample.

An ensemble machine learning classification model is integrated by many classification algorithms to improve classification performance. It was originally developed through classical methods such as Bagging [18], Boosting [19], Stacking [20], and used logical operations to make decisions on the results of multiple classifiers. Woźniak et al. believe that using an ensemble classification model is better than using a single classifier [21]. In [22], the authors proposed an optimal feature selection algorithm. The algorithm uses FVV index feature selection method to select appropriate features. Then, the algorithm removes some useless or small impact features and improves the performance of the model. The solution in [23] uses auto-encoder to recalculate features. This solution makes the classification model more sensitive to the feature values of abnormal network flows. In [24], Singh and De proposed an ensemble feature selection method. Seven feature selection methods are used to calculate the average, and features higher than the average are selected to train the model. The method improves the system performance and also reduces the overall computation time.

In [13], the authors discussed the problem of noise in the training dataset. They divided the noise handing methods into the robustness approaches, the filtering approaches and the noise consideration approaches. The robustness approaches refer to the use of classification algorithms that are not sensitive to noise. The filtering approaches are to find and eliminate the noise in the training dataset before training the classification model. The noise consideration approaches refer to considering noises in the process of training the classification model. In [25], the authors said that noise in the training dataset will affect the classification results. They proposed a system that divides the samples in the dataset into two categories: clean samples and misclassified samples. The misclassification analysis is performed on the misclassified samples to improve the classification accuracy. However, the solutions in [13, 25] are not for the botnet detection.

Recently, researchers have begun to consider noise issues in botnet flow detection. In the hybrid sampling method, [26] uses one-side selection under-sampling method to remove a small amount of noise. However, the authors only focus on the feature extraction process. In the paper, the hybrid sampling method is used to deal with the problem of unbalanced datasets in machine learning. Then, the algorithm uses CNN and BiLSTM to extract the features of network flows. Therefore, a balanced dataset was constructed for training. The above process also reduces the training time of the model and improves the classification performance. In [27], the sparse and low-rank matrix in the robust principal component analysis method is used for optimization operations to achieve the purpose of noise filtering. The model proves that it is effective for C&C communication detection.

Recently, researchers have begun to use deep learning techniques to design botnet detection solutions. In [28], the authors use convolution layer to extract flow-based features. Then, the feedforward artificial neural network is used to train the classification model. From the experimental results, the model can obtain higher detection accuracy and lower false alarm rate. In [10], the authors proposed an effective deep neural network (DNN) framework to detect cyberattacks. The solution automatically recognizes malicious characteristics by processing both host-level and network-level events. The authors evaluated the performance of the proposed solution on various NIDS and HIDS datasets. The proposed framework is highly scalable and can be enhanced to process large amounts

of data in real time. The paper in [11] proposes a deep learning botnet detection framework for IoT networks. Experimental results show that the proposed deep learning model is better than classical machine learning classifiers. The authors also analyzed the characteristics of the datasets by using the *t*-distributed stochastic neighbor embedding (*t*-SNE) visualization technique.

Machine learning algorithms are vulnerable to various adversarial attacks. In [29], the authors proposed a P2P botnet detection mechanism. It can detect noise-injected P2P botnet flows, and the detection rate can be as high as 90%. In [30], the authors proposed a technique that can detect domain name system homograph attacks and randomly generated domain names. Three adversarial attacks DeepDGA, CharBot and MaskDGS are considered to verify the robustness of the proposed model. After performing 10-fold cross-validation on four different DGA datasets, the average accuracy of the proposed model reached 0.99. In [31], the authors analyzed adversarial attacks on network intrusion detection systems. The authors randomly modify the values of four features: flow duration, send bytes, receive bytes and exchanged bytes, and update the corresponding derived features to perform the adversarial attacks. This paper provides a perspective of network intrusion detection through the analysis of adversarial disturbances. In [32], the authors proposed an adversarial attack against the deep learning NIDS in the Internet of Things environment. The author uses saliency maps to find the most critical features. The attacker only needs to modify less than 0.005% of the bytes in the malicious data packet to successfully compromise a state-of-the-art NIDS.

3 The Proposed Framework

In this section, first, we will discuss the proposed classification framework in Section 3.1. In Section 3.2, we discussed the constructions of the pre-classifier and *T*-classifier. At the end, we will discuss the construction of the *F*-classifier in Section 3.3.

3.1 The Classification Framework

The proposed classification framework is shown in Figure 1. It is an ensemble classifier that contains three classifiers: *pre-classifier*, *T-classifier* and *F-classifier*.

The *pre-classifier* is the first layer of the classification framework. It predicts the class to which the testing data belongs. Class *T* is a set of testing data that will be correctly classified by the *T*-classifier; class *F* is a set of testing data that will be misclassified by the *T*-classifier. We have to emphasize that the classification result of the pre-classifier is just a prediction. The *T*-classifier divides the testing data in class *T* into normal and botnet flows.

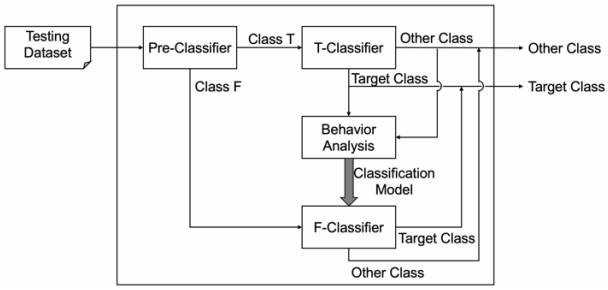


Figure 1. The classification framework

The *F*-classifier divides the testing data in class *F* into normal and botnet flows. Since the testing data in class *F* will be misclassified by the *T*-classifier, the testing data in class *F* is the noise. In the *F*-classifier, we cannot use the features or the classification algorithms used in the *T*-classifier. We will find another features and classification algorithms to train the classification model of *F*-classifier. The testing data in class *F* will be classified by the classification model trained using the classification results of testing data in class *T*.

3.2 The Construction of the Pre-classifier and *T*-classifier

In this subsection, we discuss the steps to construct the pre-classifier and *T*-classifier.

Let Ω_T be the classification algorithm to be considered in this paper. Ω_T may be support vector machine (SVM), decision tree (DT) or naïve Bayes (NB). The steps to construct the pre-classifier and *T*-classifier are shown in Figure 2. First, we use the algorithm Ω_T and the training dataset to train the *O*-classifier. Then, the training dataset is divided into classes *T'* and *F'* by the *O*-classifier. Class *T'* contains data correctly classified by the *O*-classifier; class *F'* contains data misclassified by the *O*-classifier.

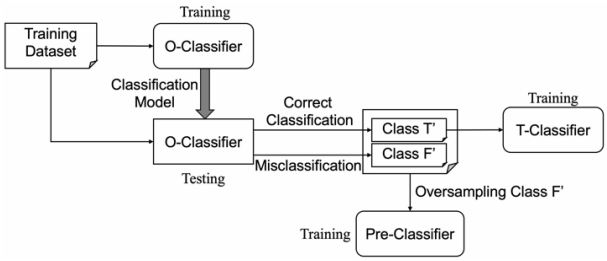


Figure 2. The steps to construct the pre-classifier and *T*-classifier

Second, the data in class *T'* and the algorithm Ω_T are used to train the *T*-classifier according to the botnet and normal labels on the data. Then, the *T*-classifier is established.

Now, let us discuss the construction of the pre-classifier. To construct the pre-classifier, we have to select a classification algorithm. Let the selected algorithm be Ω_p. From Figure 2, the training dataset is divided into classes *T'* and *F'* by the *O*-classifier.

Therefore, we can train the pre-classifier using the training dataset with labels *T'* and *F'*. The selection of algorithm Ω_p will be discussed in the next section.

In general, the number of flows in class *T'* is greater than the number of flows in class *F'*. If the number of flows in class *T'* is less than the number of flows in class *F'*, the algorithm Ω_T is not suitable for identifying botnet flows. It is better to choose another classification algorithm for the *O*-classifier.

If Ω_T correctly classifies the testing dataset, *i.e.*, the accuracy of algorithm Ω_T is very high, then class *F'* may be empty or contain only a small number of flows. In this case, improving the performance of the algorithm Ω_T is unnecessary. Without loss of generality, we assume that class *F'* is not an empty set or contains only a small number of flows. To train the pre-classifier, we have to balance the number of flows in both classes *T'* and *F'*. Because we want the *F*-classifier to only classify noise data, the over-sampling technique is applied to the class *F'*, and overfitting occurs. If class *F'* contains only a small number of flows and over-sampling techniques are used, the pre-classifier will misclassify most flows as class *F*. Then, most flows will be classified by the *F*-classifier. Therefore, this paper does not consider the classification algorithm with very high accuracy in Ω_T.

3.3 The Construction of the *F*-classifier

Class *F* contains data that will be misclassified by the *T*-classifier. Therefore, for the *F*-classifier, we cannot use the features and the classification algorithm used in the *T*-classifier.

Although the flows in class *F* are the noise of the dataset, the same category of flows in both classes *T* and *F* share the same connection properties. The source and destination ports used in these flows have locality property. In other words, the distribution of ports of flows in class *F* is similar to that in class *T*. Therefore, we will use the data classified by the *T*-classifier to construct the classification model of *F*-classifier. As shown in Figure 1, the behavior analysis module trains the *F*-classifier by using the classification results of the *T*-classifier. For the *F*-classifier, the feature vector is composed of the source and destination port numbers.

To construct the *F*-classifier, we have to select a classification algorithm. The classification algorithm will be determined by the experimental results and will be discussed in the next section.

4 Experimental Results

This section evaluates the performance of the proposed framework. All experiments were run on a Linux computer with version 4.15.0. The system was written using Python 3 with version 3.6.9. The machine learning package we used is Scikit-learn [33], version 0.22.1. The source code for the proposed classification framework is available in [34]. In the proposed framework, three classification algorithms are considered. They are the support vector machine (SVM) with RBF kernel, Gaussian naïve Bayes (NB) and decision tree (DT) classifiers. The parameters used in these algorithms are the default values of the Scikit-learn machine learning package.

In the following of this section, Section 4.1 discusses the datasets used in the experiments. Section 4.2 discusses the performance metrics. Sections 4.3 and 4.4 are experiments on feature selection and algorithm selection, respectively. Section 4.5 is the performance evaluation of the proposed framework. Then, we compare the proposed framework with the other ensemble classification methods in Section 4.6. Finally, we present a case study on the robustness of the proposed framework in Section 4.7.

4.1 The Dataset

The CTU-13 [35] dataset was used in this study. The CTU-13 dataset is a collection of 13 different botnet samples. In this paper, three scenarios CTU-Malware-Capture-Botnet-42 (CTU42), CTU-Malware-Capture-Botnet-43 (CTU43) and CTU-Malware-Capture-Botnet-50 (CTU50) are used to evaluate the performance of the proposed framework. All three scenarios are Neris botnet packet capture. The first scenario, CTU42, is used to train the classification model. The other two scenarios are used to test the performance of the proposed framework. CTU42 is also used for feature selection and algorithm selection. Two scenarios, CTU-Malware-Capture-Botnet-46 (CTU46) and CTU-Malware-Capture-Botnet-54 (CTU54), are used to evaluate the robustness of the proposed framework. These two scenarios are Virut botnet capture.

For each scenario, we collect botnet and normal flows from the complete packet capture (PCAP) format files based on the IP addresses specified in the dataset document. A flow is a sequence of bidirectional packets between specific source hosts and/or from specific destination hosts. For botnet flows, the flows of all infected IPs will be collected. For normal flows, we only collect flows from the following three IPs: 147.32.84.170, 147.32.84.134 and 147.32.84.164; it is because the documentation of the dataset indicates that other flows are less reliable.

Table 1 lists the details of these scenarios. It also shows the characteristics of botnet communication in these scenarios [35]. In order to understand the properties of the dataset, we apply *t*-SEN technology to visualize the flow records of the dataset. The *t*-SNE technology maps high-dimensional data to low-dimensional data [9, 11]. In Figure 3, the visualization of normal and botnet flow records for each scenario is shown. We can see in the figure that many normal and botnet flow records overlap. The proposed noise filtering technique will filter these noise flow records and improve the detection performance.

Table 1. CTU42, CTU43, CTU46, CTU50 and CTU54 in the CTU-13 Dataset

Scenario	Bot Name	Characteristics of botnet scenarios					Type	# of IPs	Packets	Flows
		IRC	SPAM	CF	PS	HTTP				
CTU42	Neris	√	√	√			Bot	1	206424	14428
							Normal	3	943115	27067
CTU43	Neris	√	√	√			Bot	1	177948	19440
							Normal	2	68994	8471
CTU50	Neris	√	√	√	√		Bot	10	1536686	93736
							Normal	3	1189716	27612
CTU46	Virut		√		√	√	Bot	1	31391	859
							Normal	3	125972	4623
CTU54	Virut		√		√	√	Bot	1	290492	33040
							Normal	3	623954	26898

CF: ClickFraud, PS: Port Scan

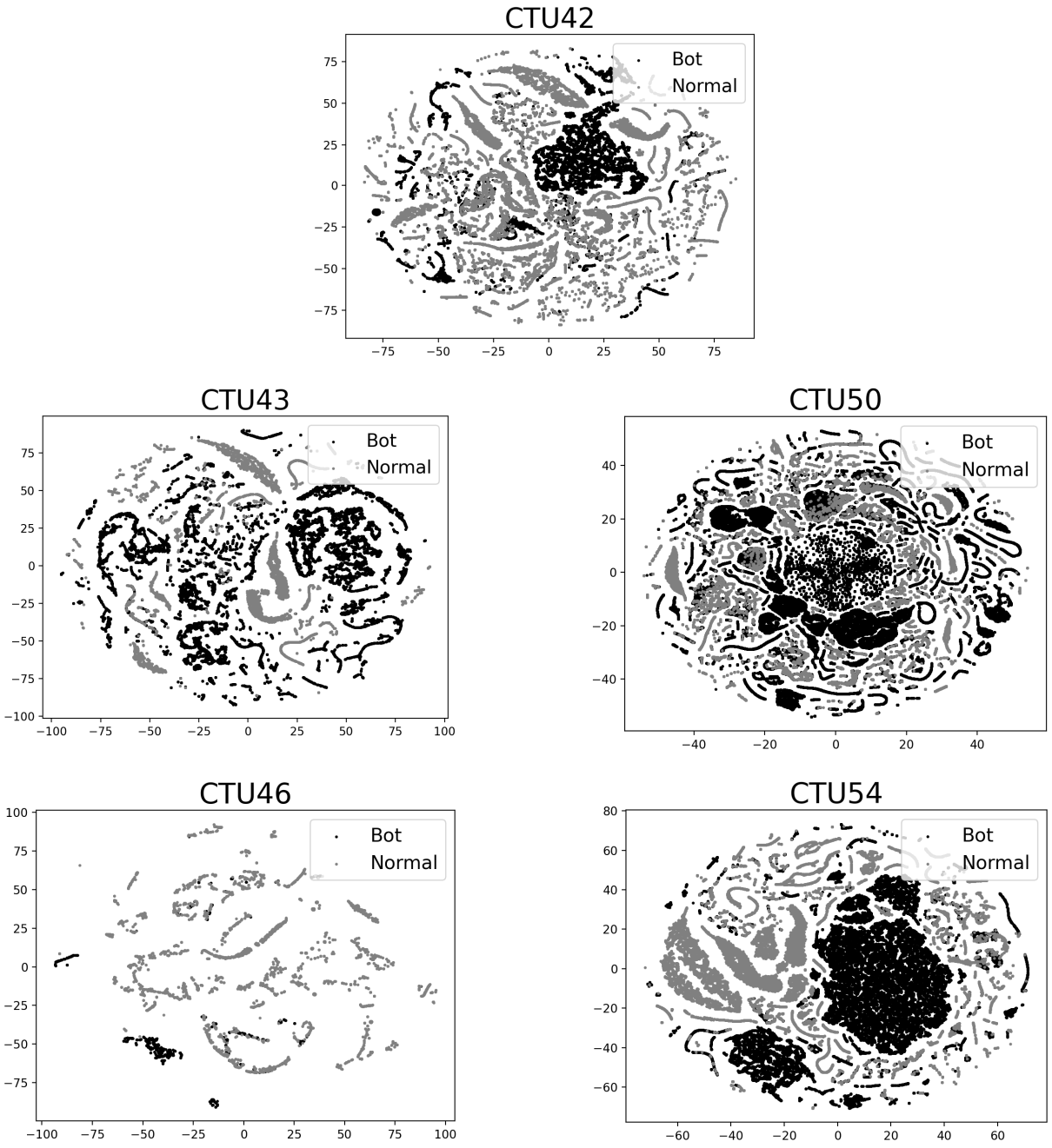


Figure 3. *t*-SNE visualization of normal and botnet flows records

4.2 Performance Metrics

In order to evaluate the proposed framework, this paper uses standard performance metrics. These metrics include *true positive rate* (*TPR*), *false positive rate* (*FPR*) and *accuracy* (*ACC*). The *true positive rate* (also known as the *detection rate*) is the ratio of the number of correctly detected botnet flows and the total number of botnet flows. The *false positive rate* (also known as the *false alarm rate*) is the ratio of the number of misclassified normal flows to the total number of normal flows. The *accuracy* is the ratio of the number of correctly detected botnet and normal flows to the total number of flows. The equations for these three metrics are *eq*. (1)-(3), where *true positive* (*TP*) is the number of botnet flows correctly classified; *true negative* (*TN*) is the number of normal flows correctly classified; *false positive* (*FP*) is the number of normal flows incorrectly classified; *false negative* (*FN*) is the number of botnet flows incorrectly classified.

$$TPR = \frac{TP}{TP + FN} \tag{1}$$

$$FPR = \frac{FP}{FP + TN} \tag{2}$$

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

4.3 The Feature Selection

In the proposed framework, we have to find the best set of features. The same features are used in *O*-classifier, pre-classifier and *T*-classifier. Then, these three classifiers will operate in the same feature space.

In order to extract features from the flow, the packets are collected as a bidirectional flow according the five-tuple (protocol plus source and destination IP addresses and ports) of each packet. In our experiments, we only consider TCP and UDP flows. We do not consider ICMP packets. Then, for each flow, we extracted 27 features. These features are categorized into five types: packet number, packet size, packet interval, outbound packets and return packets. Outbound packets are packets that are sent from the source to the destination. Return packets are packets that are sent back from the destination to the source. Then, we will select the appropriate features for the proposed framework based on the Fisher score [36]. Table 2 shows the 27 features, which are ranked according to the Fisher score calculated using the CTU42 dataset.

The first experiment is to find the appropriate number of features for each classification algorithm (SVM, NB or DT). In the experiment, we determined the appropriate number of features from the algorithm Ω_T because we did not consider the interaction between the number of features and noise. We will add features one by one in the rank listed in Table 2 and evaluate the accuracy of the classification algorithm.

Table 2. Features ranked according to their Fisher score

Rank	Feature Name	Rank	Feature Name	Rank	Feature Name
1	Frequency of Change of Direction	10	Standard Deviation of Packet Size	19	Sum of Packet Interval
2	Return-First Packet Size	11	Maximum Packet Size	20	Sum of Packet Size
3	Outbound-First Packet Size	12	Outbound-Maximum Packet Size	21	Return-Sum of Packet Size
4	Return-Minimum Packet Size	13	Outbound-Average of Packet Size	22	Minimum Packet Interval
5	Outbound-Minimum Packet Size	14	Outbound-Standard Deviation of Packet Size	23	Return-Number of Packets
6	Minimum Packet Size	15	Return-Standard Deviation of Packet Size	24	Number of Packets
7	Return-Average of Packet Size	16	Average of Packet Interval	25	Outbound-Number of Packets
8	Average of Packet Size	17	Standard Deviation of Packet Interval	26	Outbound-Sum of Packet Size
9	Return-Maximum Packet Size	18	Maximum Packet Interval	27	Number of Change of Direction

Since the normal and botnet flows in CTU42 are imbalanced, the under-sampling technique is used randomly and uniformly for the class with a large number of flows. In order to find the best parameter value, a 10-fold cross validation technique is used. Since the under-sampling technique randomly samples the data, we repeat the above process 10 times to obtain the average of the 100 results. Figure 4 shows the experimental results. For SVM, the appropriate features are the first six features. For DT, the appropriate features are all 27 features. For NB, when the number of features is three, the algorithm will obtain maximum accuracy.

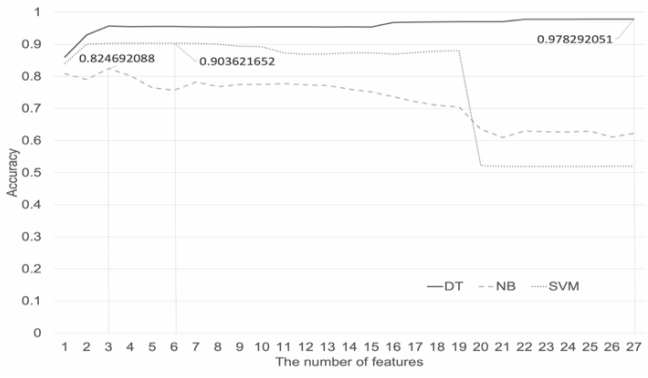


Figure 4. The number of features for classification algorithms

If the algorithm of the *O*-classifier is determined, the features selected above are used on the pre-classifier and the *T*-classifier. As mentioned in Section 3, the features of the *F*-classifier are the source port and the destination port.

From Figure 4, we also know that the accuracy of DT is 97.8% if all 27 features are used. Since the accuracy of the algorithm DT is very high, there are only a few flows left in class *F'* when training the pre-classifier. Even sometimes the class *F'* is an empty set. Therefore, in the following experiments, we do not consider the case where the algorithm of *O*-classifier is DT.

4.4 The Algorithm Selection

As mentioned in Section 3, for the proposed framework, the *T*-classifier and the *O*-classifier use algorithm Ω_T. In the following of this paper, we only consider that algorithm Ω_T is SVM or NB. In this subsection, we will find the classification algorithm of the pre-classifier and the *F*-classifier that are applicable for the two classification algorithms of Ω_T. In the experiment, the three algorithms of SVM, DT and NB were tested against the pre-classifier and *F*-classifier. A 10-fold cross validation technique is also used. Since the under-sampling technique randomly samples data, we will repeat this process 10 times to obtain the average of the 100 results.

Consider the case where the algorithm Ω_T is SVM. It can be seen from the experiment that if the pre-classifier uses SVM and the *F*-classifier uses SVM or NB, the accuracy of the proposed framework can reach up to 99.3%. The average accuracy improvement rate is 9.9%. However, the *F*-classifier will classify flows that are misclassified by the *T*-classifier. For the *F*-classifier, it is better to choose a different algorithm from the *T*-classifier. Therefore, in this case, the *F*-classifier uses NB. That is, if the algorithm Ω_T is SVM, the appropriate combination of the proposed framework for the pre-classifier, *T*-classifier and *F*-classifier is SVM-SVM-NB. The results are shown in Table 3.

Table 3. Algorithms for the pre-classifier and *F*-classifier

The algorithm Ω_T		The proposed framework				
Algorithm	Accuracy	Pre-classifier	*T*-classifier	*F*-classifier	Accuracy	Improvement rate
SVM	90.4%	SVM	SVM	NB	99.3%	9.9%
NB	83.1%	DT	NB	SVM	97.0%	16.7%

Consider the case where the algorithm Ω_T is NB. It can be seen from the experiment that if the pre-classifier uses DT and the *F*-classifier uses NB or SVM, the accuracy of the proposed framework can reach up to 97.0%, and the average accuracy improvement rate is 16.7%. However, for the *F*-classifier, it is better to choose a different algorithm from the *T*-classifier. In other words, if the *T*-classifier is NB, the appropriate combination of the proposed framework for the pre-classifier, *T*-classifier and *F*-classifier is DT-NB-SVM. The results are summarized in Table 3.

4.5 The Performance of the Proposed Framework

In this subsection, we will evaluate the performance of the proposed framework. CTU42 is used for the training dataset. Two other scenarios of Neris (*viz*., CTU43 and CTU50) are used to test the performance of the proposed framework. The under-sampling technique randomly samples the training data, so we repeat the test 100 times and take the average the 100 results. The experimental results of the proposed framework are in Table 4.

Table 4. The performance of the proposed framework

(a) The algorithm Ω_T is SVM, the proposed framework is SVM-SVM-NB

Scenario	The algorithm Ω_T			The proposed framework			Improvement rate		
	TPR	FPR	ACC	TPR	FPR	ACC	TPR	FPR	ACC
CTU42	81.3%	0.5%	93.2%	99.1%	0.4%	99.4%	21.8%	-13.5%	6.7%
CTU43	99.9%	0.7%	99.7%	100.0%	0.6%	99.8%	0.1%	-13.3%	0.1%
CTU50	81.6%	0.4%	85.7%	99.0%	0.3%	99.2%	21.3%	-28.2%	15.7%

(b) The algorithm Ω_T is NB, the proposed framework is DT-NB-SVM

Scenario	The algorithm Ω_T			The proposed framework			Improvement rate		
	TPR	FPR	ACC	TPR	FPR	ACC	TPR	FPR	ACC
CTU42	80.7%	14.6%	83.7%	97.0%	0.1%	98.9%	20.3%	-99.4%	18.1%
CTU43	96.7%	11.0%	94.4%	95.1%	0.3%	96.5%	-1.6%	-97.6%	2.3%
CTU50	77.1%	18.8%	78.0%	90.0%	0.2%	92.2%	16.8%	-99.1%	18.2%

First, we compare the accuracy of the proposed framework and the algorithm Ω_T. The proposed framework improves accuracy in all cases. Second, let us consider the false positive rate. Experimental results show that the proposed framework significantly reduces the false positive rate. In other words, the proposed framework can prevent normal flows from being classified as botnet flows. Finally, let us consider the true positive rate. For most tests, the proposed framework can also increase the true positive rate. Therefore, the improvement of the accuracy of the proposed framework is mainly affected by the reduction of the false positive rate. From Table 4(a), If the algorithm Ω_T is SVM, the true positive rate is increased by as much as 21.8%, and the false positive rate is reduced by as much as 28.2%. From Table 4(b), if the algorithm Ω_T is NB, the true positive rate is increased by as much as 20.3%, and the false positive rate is reduced by as much as 99.4%.

4.6 The Comparison with Other Methods

In this subsection, we compare the proposed framework with other ensemble network intrusion detection solutions [37-38]. In [37], the authors select features by using the *p*-value. Then, the voting technique is applied to three different classification algorithms. The authors also show five combinations of the proposed voting technique. [38] proposed a two-level classification model for network intrusion detection. The authors combine ReliefF and borderline-SMOTE algorithms to improve classification accuracy.

In the experiment, the parameters used by these algorithms are the default values of the Scikit-learn machine learning package. CTU42 is also used to train the classification models. Two other scenarios, CTU43 and CTU50, of Neris are used to test the performance of the classification models. The under-sampling technique randomly samples the training data, so we repeat the test 100 times and get the average of these 100 results.

Table 5 shows the experimental results. The two methods in group A are the proposed two classification models. The five methods in group B are the classification models in [37]; and the six methods in group C are the classification models in [38]. It can be seen from Table 5 that the proposed method, SVM-SVM-NB, has the highest accuracy. Although the false positive rate of SVM-SVM-NB is higher than that of DT-NB-SVM, its false positive rate is also lower than the methods in [37] and [38]. Therefore, the proposed method has the highest accuracy and lower false positive rate.

Table 5. The comparison with other methods

Group	Method	Scenario: CTU42			Scenario: CTU43			Scenario: CTU50		
		TPR	FPR	ACC	TPR	FPR	ACC	TPR	FPR	ACC
A	SVM_{RBF}-SVM_{RBF}-NB	**99.1%**	0.4%	**99.4%**	**100.0%**	0.6%	**99.8%**	**99.0%**	0.3%	**99.2%**
	DT-NB-SVM_{RBF}	97.0%	**0.1%**	98.9%	95.1%	**0.3%**	96.5%	90.0%	**0.2%**	92.2%
B [37]	DT-KNN-LR	96.2%	0.7%	98.2%	99.8%	0.6%	99.7%	86.1%	1.2%	89.0%
	DT-SVM_{Poly}-LR	87.7%	0.5%	95.4%	100.0%	0.7%	99.7%	82.7%	0.5%	86.5%
	DT-SVM_{RBF}-LR	88.2%	0.6%	95.5%	100.0%	0.8%	99.7%	84.3%	0.6%	87.8%
	DT-SVM_{Liner}-LR	87.4%	1.5%	94.6%	100.0%	0.9%	99.7%	85.0%	1.2%	88.2%
	AB-SVM_{RBF}-GB	96.0%	2.1%	97.3%	100.0%	1.0%	99.7%	85.8%	1.7%	88.6%
C [38]	C4.5+KNN	93.6%	7.2%	93.0%	85.9%	3.6%	89.1%	78.0%	11.0%	80.5%
	C4.5+NB	83.3%	67.1%	50.4%	79.8%	73.3%	63.7%	74.0%	63.3%	65.5%
	KNN+C4.5	96.3%	3.8%	96.3%	98.9%	2.2%	98.6%	91.7%	6.3%	92.2%
	KNN+NB	94.3%	73.3%	50.2%	95.7%	72.9%	74.9%	87.8%	73.7%	73.8%
	NB+KNN	96.5%	3.4%	96.6%	99.5%	2.0%	99.0%	93.1%	5.7%	93.4%
	NB+C4.5	93.3%	5.3%	94.2%	86.5%	2.7%	89.8%	78.9%	7.8%	81.9%

SVM_α: Support Vector Machine with kernel function α, NB: Naïve Bayes, DT: Decision Tree, KNN: *k*-Nearest Neighbor, LR: Logistic Regression, AB: AdaBoost, GB: Gradient Boosting

4.7 A Case study on the Robustness of the Proposed Framework

In this subsection, we present a case study on the robustness of the proposed framework. We will use both CTU46 and CTU54 to perform the robustness test. These two scenarios both contains flows of Virut botnets. As it was mentioned, CTU42 contains flows of Neris botnet. From Table 1, the characteristics of botnet flow records of Neris and Virut are partially different.

In the experiment, we respectively use CTU42, CTU46 and CTU54 to train the proposed methods; then, test CTU46 and CTU54. The under-sampling technique randomly samples the training data, so we repeated the test 100 times and get the average of these 100 results. In Table 6, when we training the models using CTU42 and test the Virut dataset, the true positive rate and the accuracy of the proposed model will decrease in some cases. However, the false positive rate is still very small. From the experimental results, SVM-SVM-NB is more robust than DT-NB-SVM.

Table 6. A case study of the robustness test

Training	Method	Scenario: CTU46			Scenario: CTU54		
		TPR	FPR	ACC	TPR	FPR	ACC
CTU42	SVM-SVM-NB	100.0%	0.5%	99.6%	90.1%	0.4%	94.4%
	DT-NB-SVM	79.9%	0.3%	96.6%	71.8%	0.2%	84.4%
CTU46	SVM-SVM-NB	99.9%	0.7%	99.4%	93.9%	0.5%	96.4%
	DT-NB-SVM	99.4%	0.1%	99.9%	91.7%	0.3%	95.3%
CTU54	SVM-SVM-NB	100.0%	1.0%	99.1%	99.8%	0.6%	99.6%
	DT-NB-SVM	99.8%	0.1%	99.9%	99.7%	0.3%	99.7%

Also in Table 6, when we use CTU46 and CTU54 to retrain the model respectively, the performance improves. The performance of the model is not the best if CTU46 is the training dataset. It is because CTU46 contains only a few flows (see Table 1).

Therefore, when the attack behavior changes, the performance of the proposed models may decrease. However, when we retrain the model using a dataset with new attack behaviors, the performance of the model will improve even if the training dataset contains few samples.

5 Conclusion and Future Work

This paper discusses how noise filtering technology is applied to the botnet detection problem. We proposed an ensemble classification framework for botnet detection. The major advantages of the proposed framework are as follows: (1) The proposed framework improves both the detection rate and the accuracy rate. (2) The experimental results show that the proposed framework significantly reduces the false alarm rate. (3) Even after testing different types of botnet flows, the false alarm rate is still very small. That is, the proposed framework can prevent the normal flow from being classified as a botnet flow. The result prevents the system administrator from losing confidence in the classifier.

One problem with the proposed framework is the high computational complexity. First, let us discuss the training phase. From Figure 2, to construct the proposed classifier, we have to train the *O*-classifier first. Then, use the classification results to train the pre-classifier and the *T*-classifier. Therefore, three training processes and one testing process are required to construct the proposed classifier. Now, let us discuss the testing phase. From Figure 1, a flow has to pass through two levels of classifiers to get its class. This is the same as the classification process of most ensemble classifier. However, the *F*-classifier is trained using the classification results of *T*-classifier. The delay occurs on the *F*-classifier. According to the classification results on the CTU42 dataset, only 38.4% of total flows are classified by the *F*-classifier. In other words, most testing flows will be quickly classified by the *T*-classifier.

Recently, many deep learning botnet detection solutions have been proposed, and the performance of these solutions is better than machine learning solutions. For deep learning solutions, features are inferred and optimized for the desired output automatically. Therefore, it is difficult for us to understand the impact for each feature on the solution. Bases on the results of this study, in future work, we will apply noise filtering technology to deep learning botnet detection solutions. We will also try to improve the above-mentioned problem.

References

[1] W. Z. Khan, M. K. Khan, F. T. B. Muhaya, M. Y. Aalsalem, H. C. Chao, A Comprehensive Study of Email Spam Botnet Detection, *IEEE Communications Surveys & Tutorials*, Vol. 17, No. 4, pp. 2271-2295, Fourth Quarter, 2015.

[2] A. A. Awad, S. G. Sayed, S. A. Salem, Collaborative Framework for Early Detection of RAT-Bots Attacks, *IEEE Access*, Vol. 7, pp. 71780-71790, May, 2019.

[3] A. Wang, W. Chang, S. Chen, A. Mohaisen, Delving into Internet DDoS Attacks by Botnets: Characterization and Analysis, *IEEE/ACM Transactions on Networking*, Vol. 26, No. 6, pp. 2843-2855, December, 2018.

[4] J. Canavan, The Evolution of Malicious IRC Bots, *Proceedings of Virus Bulletin Conference 2005*, Dublin, Ireland, 2005, pp. 104-114.

[5] F. Haddadi, A. N. Zincir-Heywood, Benchmarking the Effect of Flow Exporters and Protocol Filters on Botnet Traffic Classification, *IEEE Systems Journal*, Vol. 10, No. 4, pp. 1390-1401, December, 2016.

[6] I. Ghafir, V. Prenosil, M. Hammoudeh, T. Baker, S. Jabbar, S. Khalid, S. Jaf, BotDet: A System for Real Time Botnet Command and Control Traffic Detection, *IEEE Access*, Vol. 6, pp. 38947-38958, June, 2018.

[7] O. Y. Al-Jarrah, O. Alhussein, P. D. Yoo, S. Muhaidat, K. Taha, K. Kim, Data Randomization and Cluster-Based Partitioning for Botnet Intrusion Detection, *IEEE Transactions on Cybernetics*, Vol. 46, No. 8, pp. 1796-1806, August, 2016.

[8] I. Ahmad, M. Basheri, M. J. Iqbal, A. Rahim, Performance Comparison of Support Vector Machine, Random Forest, and Extreme Learning Machine for Intrusion Detection, *IEEE Access*, Vol. 6, pp. 33789-33795, May, 2018.

[9] S. Sriram, R. Vinayakumar, M. Alazab, S. KP, Network Flow based IoT Botnet Attack Detection using Deep Learning,

IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Toronto, Canada, 2020, pp. 189-194.

[10] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, S. Venkatraman, Deep Learning Approach for Intelligent Intrusion Detection System, *IEEE Access*, Vol. 7, pp. 41525-41550, April, 2019.

[11] R. Vinayakumar, M. Alazab, S. Srinivasan, Q. Pham, S. K. Padannayil, K. Simran, A Visualized Botnet Detection System Based Deep Learning for the Internet of Things Networks of Smart Cities, *IEEE Transactions on Industry Applications*, Vol. 56, No. 4, pp. 4436-4456, July-August, 2020

[12] G. L. Libralon, A. C. P. de Leon Ferreira de Carvalho, A. C. Lorena, Pre-Processing for Noise Detection in Gene Expression Classification Data, *Journal of the Brazilian Computer Society*, Vol. 15, No. 1, pp. 3-11, March 2009.

[13] B. Frenay, M. Verleysen, Classification in the Presence of Label Noise: A Survey, *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 25, No. 5, pp. 845-869, May, 2014.

[14] J. P. Anderson, Computer Security Threat Monitoring and Surveillance, James P. Anderson Company, 1980.

[15] G. Vormayr, T. Zseby, J. Fabini, Botnet Communication Patterns, *IEEE Communications Surveys & Tutorials*, Vol. 19, No. 4, pp. 2768-2796, Fourth Quarter, 2017.

[16] L. Silva, L. Utimura, K. Costa, M. Silva, S. Prado, Study on Machine Learning Techniques for Botnet Detection, *IEEE Latin America Transactions*, Vol. 18, No. 5, pp. 881-888, May, 2020.

[17] S. Wu, J. Yu, X. Fan, Research on Intrusion Detection Method Based on SVM Co-training, *2011 Fourth International Conference on Intelligent Computation Technology and Automation*, Shenzhen, Guangdong, China, 2011, pp. 668-671.

[18] L. Breiman, Bagging Predictors, *Machine Learning*, Vol. 24, No. 2, pp. 123-140, August, 1996.

[19] Y. Freund, R. E. Schapire, Experiments with a New Boosting Algorithm, *Proceedings of the 13th International Conference on Machine Learning*, Bari, Italy, 1996, pp. 148-156.

[20] D. H. Wolpert, Stacked Generalization, *Neural Networks*, Vol. 5, No. 2, pp. 241-259, 1992.

[21] M. Woźniak, M. Graña, E. Corchado, A Survey of Multiple Classifier Systems as Hybrid Systems, *Information Fusion*, Vol. 16, pp. 3-17, March, 2014.

[22] E. Zhu, Y. Chen, C. Ye, X. Li, F. Liu, OFS-NN: An Effective Phishing Websites Detection Model Based on Optimal Feature Selection and Neural Network, *IEEE Access*, Vol. 7, pp. 73271-73284, June, 2019.

[23] Y. F. Hsu, Z. He, Y. Tarutani, M. Matsuoka, Toward an Online Network Intrusion Detection System Based on Ensemble Learning, *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, Milan, Italy, 2019, pp. 174-178.

[24] K. J. Singh, T. De, Efficient Classification of DDoS Attacks Using an Ensemble Feature Selection Algorithm, *Journal of Intelligent Systems*, Vol. 29, No. 1, pp. 71-83, January, 2020.

[25] R. Pruengkarn, C. C. Fung, K. W. Wong, Using Misclassification Data to Improve Classification Performance, *2015 12th International Conference on Electrical Engineering/ Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, Hua Hin, Thailand, 2015, pp. 1-5.

[26] K. Jiang, W. Wang, A. Wang, H. Wu, Network Intrusion Detection Combined Hybrid Sampling with Deep Hierarchical Network, *IEEE Access*, Vol. 8, pp. 32464-32476, February, 2020.

[27] E. S. C. Vilaça, T. P. B. Vieira, R. T. de Sousa, J. P. C. L. da Costa, Botnet Traffic Detection Using RPCA and Mahalanobis Distance, *2019 Workshop on Communication Networks and Power Systems (WCNPS)*, Brasilia, Brazil, 2019, pp. 1-6.

[28] S. Chen, Y. Chen, W. Tzeng, Effective Botnet Detection Through Neural Networks on Convolutional Features, *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, New York, USA, 2018, pp. 372-378.

[29] P. Narang, C. Hota, H. T. Sencar, Noise-Resistant Mechanisms for the Detection of Stealthy Peer-to-Peer Botnets, *Computer Communications*, Vol. 96, pp. 29-42, December, 2016.

[30] V. Ravi, M. Alazab, S. Srinivasan, A. Arunachalam, K. P. Soman, Adversarial Defense: DGA-Based Botnets and DNS Homographs Detection Through Integrated Deep Learning, *IEEE Transactions on Engineering Management*, pp. 1-18, March, 2021.

[31] G. Apruzzese, M. Colajanni, M. Marchetti, Evaluating the Effectiveness of Adversarial Attacks against Botnet Detectors, *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*, Cambridge, USA, 2019, pp. 1-8.

[32] H. Qiu, T. Dong, T. Zhang, J. Lu, G. Memmi, M. Qiu, Adversarial Attacks Against Network Intrusion Detection in IoT Systems, *IEEE Internet of Things Journal*, Vol. 8, No. 13, pp. 10327-10335, July, 2021.

[33] Scikit-learn: machine learning in Python, [Online]. Available: https://scikit-learn.org/stable/.

[34] Source code for Noise Filtering Framework, [Online]. Available: https://github.com/BotnetNoiseFilter/NoiseFilter.

[35] The CTU-13 Dataset. A Labeled Dataset with Botnet, Normal and Background Traffic, [Online]. Available: http://www.stratosphereips.org/datasets-ctu13.

[36] C. C. Aggarwal, Data Mining: The Textbook, Springer, 2015.

[37] M. Raihan-Al-Masud, H. A. Mustafa, Network Intrusion Detection System Using Voting Ensemble Machine Learning, *2019 IEEE International Conference on Telecommunications and Photonics (ICTP)*, Dhaka, Bangladesh, 2019, pp. 1-4.

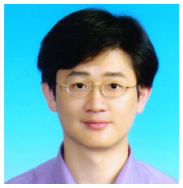
[38] J. Zhang, Y. Zhang, K. Li, A Network Intrusion Detection Model Based on the Combination of ReliefF and Borderline-SMOTE, *Proceedings of the 2020 4th High Performance Computing and Cluster Technologies Conference & 2020 3rd*

International Conference on Big Data and Artificial Intelligence, Qingdao, China, 2020, pp. 199-203.

Biographies

Tzong-Jye Liu received the Ph.D. degree in 1999 from the Department of Computer Science, National Tsing Hua University, Taiwan. After he got his Ph.D. degree, he worked several years in the computer industry in Taiwan. He was an Assistant Professor at the Feng Chia University (2004-2008), Taiwan. Dr. Liu is currently an Associate Professor at the Department of Information Engineering and Computer Science, Feng Chia University, Taiwan. His research interests include operating systems, distributed computing and network security.

Tze-Shiun Lin received his M.S. degree in the Department of Information Engineering and Computer Science from the Feng Chia University, Taiwan, 2020. Currently, he is a software engineer for work as the development of ASP.NET and WPF applications. His research interests include machine learning, data mining, and e-learning.



Ching-Wen Chen received a M.S. degree in the Department of Computer Science from the National Tsing-Hua University, Taiwan, 1995. He obtained his Ph.D. in Computer Science and Information Engineering from the National Chiao-Tung University, Taiwan, 2002. He was an Assistant Professor at Chaoyang University of Technology (2002-2005) and Feng Chia University (2005-2007), Taiwan and an Associate Professor at the Feng Chia University (2007-2013), Taiwan. Currently, he is a Professor at the Department of Information Engineering and Computer Science at the Feng Chia University, Taiwan. His research interests include computer architecture, parallel processing, embedded systems, mobile computing, and wireless sensor network.