TUNING HYPERPARAMETERS OF SELF-ORGANIZING MAPS IN COMBINATION WITH K-NEAREST NEIGHBORS FOR IOT MALWARE DETECTION

Huu Noi Nguyen[1], *Tuan Hao Hoang*[1], *Van Loi Cao*[1], *Nguyen Ngoc Tran*[1*]
DOI: 10.56651/lqdtu.jst.v12.n1.654.ict

Abstract

In the Internet of Things, sensor devices often generate massive sensory data across multiple domains and applications. Identifying IoT malware from a huge amount of such IoT data is often a challenging task. In our previous studies, analytic techniques were applied to reduce dimensionality and discover valuable information from the original data. Particularly, the Self-organizing Maps (SOM)-based classifier with an AutoEncoder is used to create an end-to-end IoT malware detection model. However, the SOM-based classifier has a constraint that new instances may be incorrectly classified if they are mapped into unlabelled neurons in the SOM map. To address this issue, in this study, a novel hybrid between SOM-based classifier and well-known classification algorithms like K-Nearest Neighbors, Support Vector Machine, Softmax, Random Forest. In this hybrid, classification methods will help to correctly assign labels for instances mapped into the unlabeled neurons. In addition, this article investigates hyperparameter optimization methods for optimizing SOM hyperparameters. Our proposed methods were tested on the NBaIoT dataset with various experimental settings. Experimental results illustrate that SOMKNN often performs better than stand-alone techniques, including the SOM classifier.

Index terms

SOM, KNN, IoT, hyperparameter optimization, malware detection.

1. Introduction

Internet of Things (IoT) has played an essential role in human life. However, IoT security faces new challenges due to the fast growth and a large-scale volume of IoT systems [1]. IoT attacks have become more frequent and sophisticated, targeting different layers and components of IoT systems, such as devices, gateways, servers, and networks. One of the notable attacks is named as Mirai. The attack is a special kind of botnet exploiting vulnerabilities on IoT devices to launch large-scale DDoS attacks [2].

[1]Institute of Information and Communication Technology, Le Quy Don Technical University
[*]Corresponding author, email: ngoctn@lqtdu.edu.vn

Anomaly detection is a task that can benefit from unsupervised learning techniques. These algorithms take unlabelled input data and try to find correlations between them. They group data into clusters based on their similarity. For example, clustering techniques for anomaly detection can separate normal data from anomalies based on the size and density of the clusters. Anomalies are data points that deviate significantly from normal patterns. They tend to form small and sparse clusters, while normal data often construct larger and denser regions. K-means [3] and SOM [4] are common unsupervised learning methods applied for identifying anomalies. These techniques can also combine with deep learning techniques, such as AutoEncoders (AEs), to improve their performance.

Previous works have used SOM [5] for anomaly detection in different domains. Allahdadi et al. [6] applied Hidden Markov models on SOM to detect anomalies in wireless networks. In [7], [8], the authors explored unsupervised, data-driven methods, such as hierarchical SOM, building anomaly detection models for real-world smart meter data. Li et al. [9] proposed a model that employs SOM and topological memory based on multi-scale features to capture normal characteristics.

In [10], Nguyen et al. employed hybrid AEs and SOMs to detect IoT malware. In the hybrid, AEs were employed to construct a new feature space for the original data, while SOMs mapped the input data onto a neuronal map. The neurons were then grouped into labeled clusters, referred to as a labeling-map, which could be used to classify new data. In a subsequent publication [11], the authors improved their methodology by incorporating a denoising step before feeding the data into the model. This improved method referred to as DAESOM, yielded superior results.

However, the previous studies remain with limitations related to the classification phase of SOM models. In SOM-based classification models, the voting method is commonly used to assign class labels for instances at each point in the SOM map. Once the labeling-map is created (from the training phase), the test instance is mapped to specific neurons in the network, then assigned a label with the labels of the trained neurons. If the instance is mapped into an unlabeled neuron in the SOM map, then the label of the majority class is assigned to that neuron. This can lead to the misclassification of some points in some regions of the SOM map if the majority class overwhelms others in those regions. Therefore, in this research, we propose the combination of SOM and KNN to overcome this limitation. The neurons' labels are determined in the test phase using K-Nearest Neighbors algorithm if no suitable labels are found.

The contributions of this article are as follows:

- Introducing a hybrid of SOM and KNN to enhance the ability to detect anomalies in IoT. SOM categorizes instances into the same groups, while the KNN measures the distance between incoming instances and the neurons in the map.
- Investigating the hyperparameters of SOM and optimizing them to find optimal values, thereby evaluating their impact on the SOM model.

- Testifying the model for unknown IoT attack detection on different scenarios. The results illustrate that the hybrid model of SOM and KNN has the ability to identify unknown attacks.

The rest of this article consists of five parts as follows: Part 2 is an introduction to the fundamentals of SOM. In Part 3, we present a detailed description of our proposed model. In Part 4, we report on our experimental methodology, including the dataset, parameter settings, and evaluation metrics utilized. In Part 5, we present and discuss our results. Finally, in Part 6, we summarize our findings and suggest directions for future research.

2. Self-organizing map

2.1. Self-organizing map

Self-organizing Map is a special case of neural networks. SOM can group similar instances on a map in which each neuron is allocated in fixed positions and adjacent to other neurons. SOM was originally proposed by Kohonen [12] and utilized in an unsupervised learning manner. They can map data from the original feature space into the mapping space of SOM (typically two-dimensional). Fig. 1 illustrates SOMs.

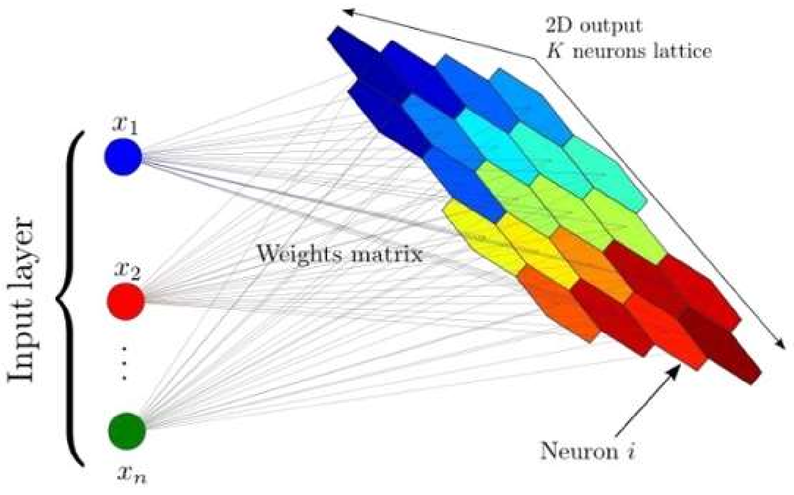


Fig. 1. The Self-Organizing Maps.

Unlike other artificial neural networks, SOM uses competitive learning instead of error-correction learning [13]. Competitive learning is a way of training a single-layer neural network without supervision or labels. A neighborhood function helps SOMs maintain the topological properties of the input data. This means that similar inputs can be mapped into nearby neurons on the map.

SOM networks can be used for various tasks such as identification, data clustering, text prediction, data mining. They can deal with various types of data, such as sound, image, and text. Section 2.2 will explain how this algorithm works in more detail.

2.2. SOM Algorithm

The winning neuron on the trained SOM is determined by using Euclidean distance, as shown in Eq. 1.

$$d_j(x) = \sqrt{\sum_{i=1}^{d}(x_i - w_{ji})^2} \tag{1}$$

In this equation, x is a data example, w_j is the weighting vector for the neuron j, and d is the number of features.

Once the winning neuron has found, its weighting parameters will be updated to make it closer to the data point. Then, a neighborhood map around it is defined. The next step is to update the weights of the neighboring neurons in order to put them closer to the winning neuron. A function, typically the Gaussian function 2, can be employed for identifying the neighborhood.

$$h_{j,i} = exp(-\frac{d_{j,i}^2}{2\sigma^2}) \tag{2}$$

where $h_{j,i}$ refers to the neighborhood of the current winning neuron i and the previous one j, $d_{j,i}$ is the distance between these neurons, and σ is a bandwidth parameter to the neighborhood.

Eq. 3 shows how the weight vector w_j is updated by adding a fraction of the difference between the input instance x and the current weighting vector. The η parameter represents the learning rate.

$$\Delta w_j = \eta h_{j,i}(x - w_j) \tag{3}$$

Eq. 4 is a formula to calculate the weighting vector for the next step of $(t + 1)$.

$$\Delta w_j(t + 1) = w_j(t) + \eta h_{j,i}(x - w_j(t)) \tag{4}$$

The details of how to train SOMs are presented in Algorithm 1. It starts with randomly initializing the weight matrix with PCA. Then, the nearest neuron is extracted from the grid Ω. Next, it updates the weights of all neighboring neurons based on Eq. 4. The process repeats until the SOM converges and the weight distribution in the map matches the input distribution. This means that each input is mapped to an appropriate position in the SOM network.

In Algorithm 1, X refers to the input dataset, q and a represent the size and dimensionality of X, l is the number of the data classes and n is the number of neurons on the resulting SOM map.

Algorithm 1 Training SOM

```
$e$: a number of epochs
Input: $X = [q, (a + l)]$
Output: $W = [n, a]$

  for $i \leftarrow 1$ to $e$ do
    Initialize weight matrix $W$ randomly or using PCA
    for $j \leftarrow 1$ to $q$ do
      $o(x_j) = argmin_k ||x_j - w_k||$, $k \in \Omega \leftarrow$ Find the nearest neuron
      $w_k(i+1) = w_k(i) + \eta(i)h_{k,o(x_j)}(i)(x_j(i) - w_k(i)) \leftarrow$ Update the weights of the
      neuron and its neighbors
    end for
  end for
  return $W$
```

3. SOMKNN model

3.1. SOMKNN model

To use SOM for recognition, the training dataset is mapped onto the SOM map, which has labels based on the existing classes in the dataset. This is called a labeling-map. Euclidean distance is often used to measure the similarity between the input instances and the map's weighting vectors. This determines the best match unit (BMU) and a ranked list of BMUs for each input pattern. If an input vector belongs to a certain class, it can also label the SOM neurons accordingly. Fig. 2 shows an example of how BMUs represent instances.

To classify an instance x_i, we first represent each class by a binary vector v_i. The value of $v_{i,j}$ is 1 if the instance x_i belongs to class c_j, and 0 otherwise. Then, we map the instance into its nearest neuron and obtain the prototype vector (denoted by $\bar{v}$). The vector $\bar{v}$ is the average of the training class vectors that are mapped to the same neuron as shown in Eq. 5. This equation uses S_n to represent all the training examples that are mapped to neuron n. It also uses $S_{n,j}$ to represent the instances that belong to class c_j and are mapped to neuron n.

$$\bar{v}_{n,j} = \frac{S_{n,j}}{S_n} \tag{5}$$

We described the SOM-based classification algorithm in our previous studies [10] [11]. By comparing the $\bar{v}_{n,j}$ with the *threshold*, instance n can be assigned by the class c_j. We set the *threshold* to 0.5 in our experiments. Then we assign 1 to the positions that have values equal or greater than 0.5, and 0 to the rest. We use a simple method to classify a new sample based on the label of its nearest neuron. We assign a label c to a neuron if a majority of the samples mapped to the neuron have the same label c.

If a sample is mapped to an unlabelled neuron, we will assign it with the label of the major class in the dataset.

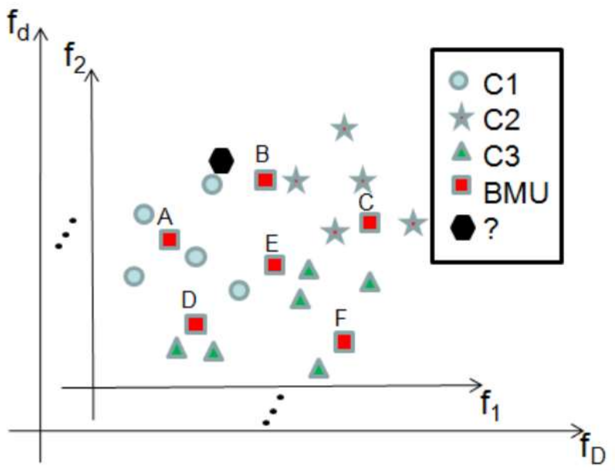


Fig. 2. The figure illustrates how SOM represents a dataset with 3 classes using BMUs. The features are plotted on the axes and the f_d indicates the high dimensionality of the input samples. The hexagon labeled with "?" is a new input vector that the SOM needs to classify.

A drawback of this approach is that a new object may get the label of the majority class if it is mapped to a neuron without a label, even though it belongs to the minority class. To address this issue and enhance classification performance, we proposed to use KNN to determine the class of that neuron. In this approach, SOM serves as a preprocessing step for the KNN classifier. SOM reduces the data dimensionality and passes the labels to KNN. Fig. 3 illustrates the main idea of this approach. As the data space after SOM is relatively small (depending on the size of the SOM plane), KNN is quick and effective as illustrated in the experimental results illustrated in Part 5.

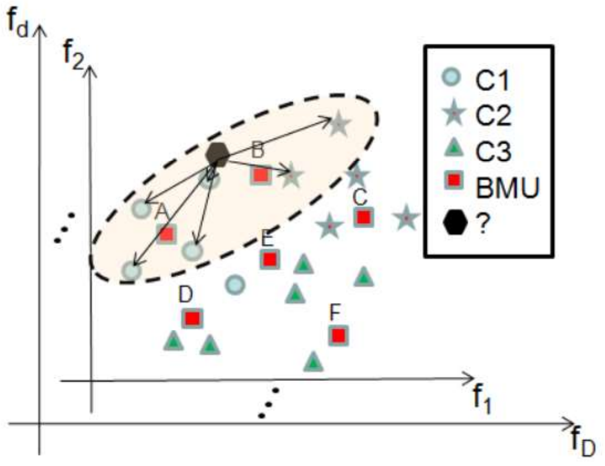


Fig. 3. Hybrid SOM and KNN (SOMKNN) for classification.

Algorithm 2 The classification algorithm using SOM

```
Input: $X^{train} = [q, (a + l)],\ W = [n, a]$
Output: $P$
  for $j \leftarrow 1$ to $m$ do
    Find the nearest neuron $o(x_j^{test})$ on the grid $\Omega$ by minimizing the distance to $x_j^{test}$
    Retrieve the training examples $T$ mapped to $o(x_j^{test})$
    Compute the average label vector $\bar{v}_j$ from $T$
    Add $\bar{v}_j$ to $x_j^{test}$
    Set $p_j$ as $\bar{v}_j$
    Compare $p_j$ to the threshold
    if $p_j > threshold$ then
      Assign the class $c_j$ that corresponds to $p_j$
    else
      Use KNN to find the label of $k$ closest neighbors
    end if
  end for
  return $P$
```

The Algorithm 2 describes the steps of how to assign labels for querying instances using SOMKNN.

- Find the closest neuron to the input on the grid Ω;
- Get the training data points that are associated with that neuron;
- Calculate the average vector (prototype vector) of those data points;
- Check if the average vector meets the *threshold* criterion;
- Determine the class label based on the average vector;
- If no class label is found, use KNN to identify the nearest neighbors of the input;
- Assign the label found by KNN to the neuron.

In Algorithm 2, q and a are the size and the dimensionality of the training data, respectively, l is the number of data classes, m is the size of the testing data, W and P are the weight and prediction matrix for each.

3.2. SOM hyperparameter optimization

Optimization is the search for the best solution among a set of possible choices that either maximizes or minimizes a given objective function for a problem [14]. In real-world scenarios, optimization problems include constraints that limit feasible solutions.

These problems can be expressed using mathematical notation as follows:

$$\begin{aligned} &\min_x f(x), \\ &subject\,to \\ &g_i(x) \leq 0, i = 1, 2, ..., m \\ &h_j(x) = 0, j = 1, 2, ..., p \\ &x \in X \end{aligned} \tag{6}$$

where $g_i(x), (i = 1, 2, ..., m)$, $h_j(x), (j = 1, 2, ..., p)$ are constraint functions, while X is the set of possible values of x.

There are two groups of parameters in the proposed model:

- Model parameters: they can be are updated throughout the training phrase. For example, weights and bias.
- Hyperparameters: they can only be chosen beforehand and their values do not change during the training phrase. For instances learning rate, the size of hidden layers, the number of layers.

In SOMs, there are three hyperparameters to optimize, including n - size of the SOM grid, η - learning rate, and σ - the bandwidth of the neighborhood function (see Eq. 2). To search for the optimized values for hyperparameters, different optimization algorithms are utilized [15]. In the research [11], we investigated the optimization methods, such as Random Search [16], Tree-structured Parzen Estimators (TPE) [15], [16], Adaptive TPE [15], Annealling [16].

In common, the steps of the hyperparameters optimization process are listed as below:

- Establish an objective function to minimize;
- Determine a search space [1] [17];
- Create a database to store the current; evaluation values of the search process;
- Create the search algorithm;
- Reduce the search space;
- Return the best values of hyperparameter as the final solution.

In experiments, we use the HyperOpt [2] (Bergstra et. al. [18]) to optimize the SOM hyperparameters.

[1]**Search Space**: In optimization, a search space or feasible region is a set of values in which the objective function will be optimized [17]. The search space D of x can be expressed by:

$$D = \{x \in X | g_i(x) \leq 0, h_j(x) = 0\} \tag{7}$$

[2]https://github.com/hyperopt/hyperopt

Table 1. The description of NBaIoT dataset

D	DeviceName	Type	Benign	Gafgyt	Mirai
1	Danmini_Doorbell	Doorbell	49548	652100	316650
2	Ecobee_Thermostat	Thermostat	13113	512133	310630
3	Ennio_Doorbell	Doorbell	39100	316400	
4	Philips_B120N10_Baby_Monitor	Monitor	175240	312273	610714
5	Provision_PT_737E_Security_Camera	Camera	62154	330096	436010
6	Provision_PT_838_Security_Camera	Camera	98514	309040	429337
7	Samsung_SNH_1011_N_Webcam	Webcam	52150	323072	
8	SimpleHome_XCS7_1002_WHT_Security_Camera	Camera	46585	303223	513248
9	SimpleHome_XCS7_1003_WHT_Security_Camera	Camera	19528	316438	514860

4. Experiments

This part will describe experiments for evaluating the performance of the SOMKNN model for IoT attack classification. We use the NBaIoT dataset, which contains nine types of IoT devices, as explained in the following subsection.

We conducted experiments in various settings to evaluate the performance and accuracy of the proposed methods for anomaly detection. We also compared the results of different hyperparameter optimization algorithms. The following scripts were used to run the experiments:

- *Improve the SOM-based method using labels-map.* This experiment evaluates the performance of SOM and SOMKNN for anomaly detection.
- *Hyperparameter optimization.* We compare different hyperparameter optimization algorithms to find the best one.
- *Attack detection.* We use the SOM/SOMKNN model with the best hyperparameter optimization algorithm to detect attacks.
- *Unknown attack detection.* We test the ability of the combined model to detect unknown attacks.

4.1. Datasets

Y. Meidan et al. [19] introduced the NBaIoT dataset [3] which consists of data samples from nine IoT devices belonging to four groups: doorbell (Danmini and Ennio), thermostat (Ecobee), monitor (the baby monitor Philips B120N10), and camera/webcam (security cameras Provision PT-737E, SimpleHome XCS7-1002-WHT, Simple Home XCS7-1003-WHT and Provision PT-838, Webcam Samsung SNH-1011-N). Each sample has 115 features extracted by Kitsune [20]. Detailed information about device types and attacks is shown in Table 1.

We used all devices (D1-D9 in Table 1) as separate datasets for training and testing with the scripts. The data were cleaned to remove any imbalance or missing values before feeding them to the algorithms. The data is split into 80% for training and 20% for testing.

[3]https://archive.ics.uci.edu/ml/datasets/detection_of_IoT_botnet_attacks_N_BaIoT

4.2. Evaluation metrics

We use the Area Under the Curve (AUC) as the evaluation metric to measure the performance of classifiers. First, the formulas for calculating the true positive rate (TPR) and false positive rate (FPR) are as follows:

$$TPR = \frac{TP}{TP + FN} \tag{8}$$

$$FPR = \frac{FP}{FP + TN} \tag{9}$$

The ROC curve represents the TPR versus the FPR at different thresholds. As the threshold for classifying an item as positive decreases, both the FPR and the TPR increase. The AUC is the area under the ROC curve. It measures how well the model can distinguish between classes at any threshold. AUC values close to 0.5 indicate no discrimination, 0.7 to 0.8 are acceptable, 0.8 to 0.9 are excellent and above 0.9 are outstanding.

5. Results and analysis

In this part, we discuss the performance of the models evaluated in Part 4. We also provide some insights into the obtained results.

We used Python with Keras, Scikit-learn and Minisom frameworks to implement the experiments. The experiments ran on a computer with Ubuntu 18.04 LTS, Intel® Core i7 930 CPU, and 18 GB memory.

We aim to classify the traffic as benign or attack. Table 1 shows the different attack types in the dataset. We group all the attack types (syn, ack, dos, ...) into one class (called attack class). Thus, we have two classes: benign and attack. We conducted experiments on *Hyperparameter optimization*, *Attack detection*, and *Unknown attack detection*. The main results are illustrated in the subsections below.

5.1. Hyperparameter optimization

In this part, we present the experiments for comparing the hyperparameter optimization algorithm. The process contains the following steps:

- Optimize the SOM parameters using different algorithms.
- Evaluate the SOM-based classification algorithm on attack and unknown attack detection with the optimized parameters to determine the most suitable for SOM.

We used devices D1, D3, D5, D6, D8 for hyperparameter tuning, following previous studies [10] [11]. This resulted in five datasets for the experiments. The tuned hyperparameters were applied to both Gafgyt and Mirai in the training processes.

Table 2. Hyperparameter tuning results

Device	Hyper-parameter	Algorithm				Algorithm			
		Train on Gafgyt				Train on Mirai			
		Rand	TPE	ATPE	Anneal	Rand	TPE	ATPE	Anneal
D1	η	3.27	3.21	0.28	2.56	4.39	0.93	3.41	3.19
	σ	7.80	7.06	9.66	8.76	8.81	8.84	6.11	7.72
	n	46.81	40.08	34.37	37.79	43.99	41.23	38.34	43.18
D3	η	4.35	1.02	1.41	0.63				
	σ	6.96	5.80	5.93	6.82				
	n	43.34	47.44	47.54	39.30				
D5	η	2.98	2.47	3.89	1.11	2.06	3.72	2.13	1.38
	σ	7.06	9.23	9.23	8.66	6.77	7.53	9.42	9.39
	n	44.09	39.53	48.94	36.84	28.02	46.59	49.94	44.51
D6	η	0.60	3.11	3.51	4.61	1.11	4.32	2.30	1.72
	σ	8.97	5.66	8.79	8.86	7.34	8.38	8.03	6.25
	n	44.22	46.26	49.55	47.84	46.88	47.51	44.98	42.37
D8	η	2.95	3.94	1.35	4.74	0.30	4.48	2.41	2.71
	σ	9.53	9.03	9.67	6.33	9.58	6.56	8.11	5.23
	n	48.25	40.36	39.58	41.47	35.11	48.41	49.74	37.34

Table 3. The performance of the SOM-based classifiers with hyperparameter tuning

Device	Test	Algorithm				Algorithm			
		Rand	TPE	ATPE	Anneal	Rand	TPE	ATPE	Anneal
		Train on gafgyt				Train on Mirai			
D1	Gafgyt	0.994	0.998	0.997	0.998	0.978	0.996	0.994	0.881
	Mirai	0.777	0.662	0.796	0.709	0.995	0.998	0.998	0.994
D3	Gafgyt	0.995	0.995	0.994	0.994				
D5	Gafgyt	0.998	0.997	0.998	0.997	0.985	0.961	0.994	0.985
	Mirai	0.684	0.822	0.675	0.723	0.992	0.991	0.995	0.991
D6	Gafgyt	0.996	0.996	0.998	0.997	0.965	0.968	0.981	0.891
	Mirai	0.679	0.682	0.681	0.683	0.995	0.999	0.996	0.998
D8	Gafgyt	0.993	0.996	0.996	0.995	0.958	0.652	0.981	0.960
	Mirai	0.778	0.675	0.670	0.645	0.995	0.989	0.995	0.992

We skip device D3 for Mirai in the tuning process because it has no Mirai attacks. We tune three hyperparameters η - learning rate, σ - the neighborhood function, and n - the map size for the SOM model. Then, we train the SOM model with the tuned values and use its output for KNN to detect anomalies.

After obtaining the values for hyperparameters, these values are used in pure-SOM based classification for both the known and unknown attack detection. In the experiments, devices $D1, D3, D5, D6,$ and $D8$ are employed for training and evaluating. We use the same/unknown attack type on the same device for training and testing. Table 3 shows that the ATPE algorithm gave the best result. Therefore, we used ATPE to optimize the hyperparameters for SOM.

The ATPE algorithm is also used for optimizing hyperparameters of other classifiers such as SVM, Decision Tree (DT), Random Forest (RF), and XGBoost (XGB). The results are shown in Table 4.

Table 4. Hyperparameters of models (SVM, RF, DT, XGB) after training

Classifier	Hyperparameters	Optimal values
SVM	C, kernel, degree, probability	{'C': 91, 'degree': 2, 'kernel': 2, 'probability': 0}
RF	max_depth, min_sample_split, max_leaf_nodes, min_samples_leaf, n_estimators, max_sample, max_features	{'max_depth': 17.94 'min_samples_leaf': 1.60, 'min_samples_split': 2.62, 'n_estimators': 120.07}
DT	max_depth, max_features, max_leaf_nodes, min_samples_leaf, min_weight_fraction_leaf, splitter	{'max_depth': 5, 'max_features': 'auto', 'max_leaf_nodes': 40, 'min_samples_leaf': 2, 'min_weight_fraction_leaf': 0.1, 'splitter': 'random'}
XGB	x_colsample_bylevel, x_colsample_bytree, x_learning_rate, x_max_depth, x_min_child_weight, x_n_estimators, x_subsample	{'x_colsample_bylevel': 14, 'x_colsample_bytree': 9, 'x_learning_rate': 14, 'x_max_depth': 2, 'x_min_child_weight': 15, 'x_n_estimators': 8, 'x_subsample': 14}

5.2. Attack detection

This subsection describes how we evaluated SOM/SOMKNN and compared it with other algorithms such as Random Forest (SOMRF), Support Vector Machine (SOMSVM), Softmax (SOMSM), KNN, SVM, RF, DT and XGBoost (XGB) for attack detection. We used devices D1-D9 for training and testing all algorithms.

We use the hyperparameters tuned in the previous part to train and test SOM-based anomaly detection models on Gafgyt and Mirai datasets. We use all devices (D1-D9) except D3 and D7, which only have Gafgyt data and no Mirai attack data.

Table 5 shows the test results. We compare the performance of 10 algorithms (SOM, SOMKNN, SOMSVM, SOMSM, SOMRF, KNN, SVM, RF, DT, XGB) using precision to 3 decimal places. For Gafgyt malware, SOMKNN outperforms the others on devices D1, D2, D3, D7 and D9. XGB does better on devices D4, D5, D6 and D8. SOM-based algorithms have the same output on devices D1, D3, D4, D6 and D7. For Mirai malware, SOMKNN beats the others on devices D1, D2, D4, D8 and D9. SOM-based classifiers have the same results on devices D1, D4, D5, D8 and D9. SOMKNN also matches the other SOM-based algorithms on all devices.

In this experiment, we visualized the data after training the SOM algorithm. As seen in Fig. 4b, benign and attack data are separated after applying SOM. SOM assigns each record to the region that matches its type. This helps the classification algorithms to detect anomalies better.

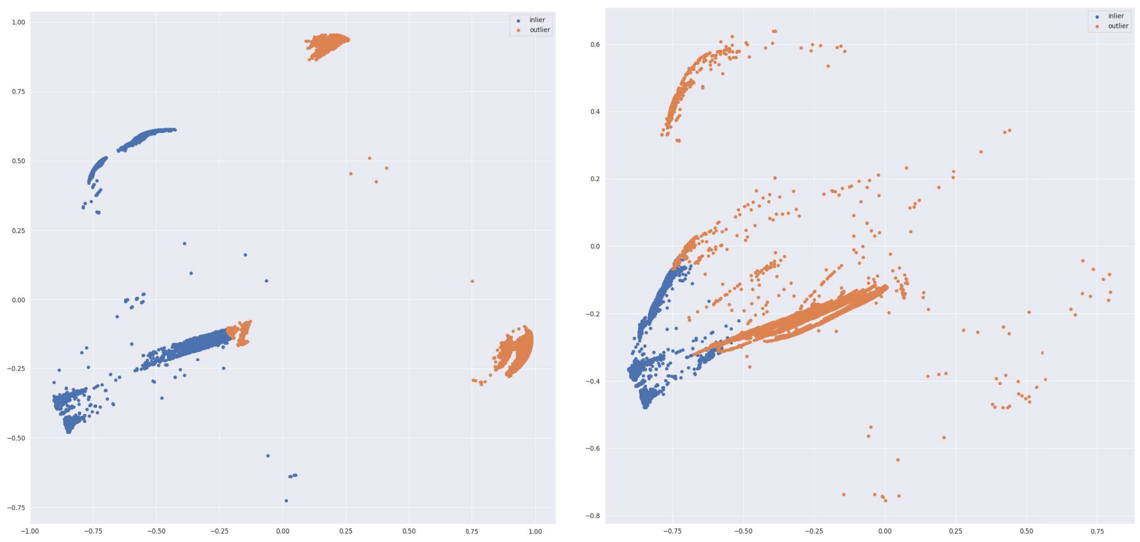
5.3. Unknown attack detection

One of the requirements in network security is the ability to identify novel attacks. The same models with optimized hyperparameters are evaluated for unknown anomaly detection. We train on Gafgyt and test on Mirai, and vice versa. We exclude devices D3 and D7 because they have no Mirai data.

Table 6 shows the results when training on Gafgyt and evaluating on Mirai, and vice versa. The table shows that the performance of the hybrid SOM and KNN is promising.

Table 5. The results of attack detection

Train/Test	**Model**	**Device**								
		D1	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**D8**	**D9**
Gafgyt/Gafgyt	**KNN**	0.995	0.997	0.989	0.988	0.988	0.988	0.988	0.988	0.988
	SVM	0.530	0.528	0.528	0.528	0.529	0.530	0.527	0.529	0.528
	RF	0.995	0.997	0.996	0.998	0.981	0.992	0.996	0.992	0.998
	DT	0.995	0.997	0.995	0.998	0.996	0.997	0.997	0.995	0.997
	XGB	0.995	0.998	0.995	0.999	0.998	0.998	0.997	0.997	0.998
	SOM	0.996	0.998	0.995	0.999	0.997	0.998	0.997	0.993	0.998
	SOMSVM	0.996	0.998	0.995	0.999	0.997	0.998	0.997	0.993	0.998
	SOMSM	0.996	0.998	0.995	0.999	0.997	0.998	0.997	0.993	0.998
	SOMRF	0.996	0.998	0.995	0.999	0.997	0.998	0.997	0.993	0.998
	SOMKNN	**0.996**	**0.998**	**0.996**	**0.999**	0.997	**0.998**	**0.997**	0.993	**0.998**
Mirai/Mirai	**KNN**	0.995	0.989		0.982	0.989	0.985		0.992	0.980
	SVM	0.530	0.501		0.500	0.500	0.501		0.500	0.500
	RF	0.999	0.998		0.998	0.998	0.997		0.995	0.993
	DT	0.998	0.997		0.997	0.997	0.996		0.994	0.992
	XGB	0.999	0.998		0.998	0.998	0.998		0.995	0.993
	SOM	0.999	0.998		0.998	0.997	0.997		0.995	0.993
	SOMSVM	0.999	0.998		0.998	0.997	0.997		0.995	0.993
	SOMSM	0.999	0.998		0.998	0.997	0.997		0.995	0.993
	SOMRF	0.999	0.998		0.998	0.997	0.997		0.995	0.993
	SOMKNN	**0.999**	**0.998**		**0.998**	0.997	0.997		**0.995**	**0.993**



(a) On the same type of attack *(b) On the other attack type*

Fig. 4. Visualization of inliers and outliers.
Inliers are shown in blue and outliers are shown in orange, respectively.

Table 6. The results of unknown attack detection

Train/Test	**Model**	**Device**						
		D1	**D2**	**D4**	**D5**	**D6**	**D8**	**D9**
Gafgyt/Mirai	**KNN**	0.770	0.771	0.740	0.712	0.695	0.725	0.735
	SVM	0.500	0.500	0.500	0.500	0.500	0.500	0.500
	RF	0.810	0.656	0.761	0.667	0.732	0.783	0.641
	DT	0.820	0.730	0.790	0.638	0.849	0.790	0.632
	XGB	0.820	0.780	0.790	0.680	0.899	0.795	0.651
	SOM	0.826	0.782	0.833	0.681	0.684	0.713	0.655
	SOMSVM	0.826	0.782	0.833	0.681	0.684	0.713	0.655
	SOMSM	0.826	0.782	0.833	0.681	0.684	0.713	0.655
	SOMRF	0.826	0.782	0.833	0.681	0.684	0.713	0.655
	SOMKNN	**0.826**	**0.805**	**0.833**	**0.681**	0.684	0.713	**0.655**
Mirai/Gafgyt	**KNN**	0.635	0.657	0.629	0.635	0.634	0.648	0.631
	SVM	0.500	0.500		0.501		0.500	
	RF	0.678	0.678	0.656	0.663	0.670	0.673	0.680
	DT	0.685	0.680	0.677	0.684	0.687	0.684	0.681
	XGB	0.686	0.678	0.636	0.610	0.678	0.676	0.610
	SOM	0.980	0.996	0.968	0.968	0.642	0.973	0.951
	SOMSVM	0.980	0.996	0.968	0.968	0.642	0.973	0.951
	SOMSM	0.980	0.996	0.968	0.968	0.642	0.973	0.951
	SOMRF	0.980	0.996	0.968	0.968	0.642	0.973	0.951
	SOMKNN	**0.980**	**0.996**	**0.968**	**0.968**	0.642	**0.973**	**0.952**

SOMKNN is also compared with other classifiers, such as SOM, SOMSVM, SOMSM, SOMRF, KNN, SVM, RF, DT, and XGB in terms of identifying unknown anomalies. The classifiers are first trained on Gafgyt and evaluated on Mirai, and then vice versa in the second case. The results are shown in Table 6. It can be seen that SOMKNN often outperforms the other stand-alone methods on most devices (D1, D2, D4, D5, and D9), and hybrid models in some cases (D2 and D9). In the first row, XGB performs better than others on D6 and D8, while produces poorly performance on the second. The results demonstrate that SOMKNN is generally more effective than the others.

Table 6 shows that training on Gafgyt and testing on Mirai gives lower accuracy than the opposite case. This is because Gafgyt and benign may have some common features, while Mirai is more distinct from benign. Both Gafgyt and Mirai are botnets that infect IoT devices and create DDoS traffic. Gafgyt is also a variant of Mirai, but it focuses on attacking the target rather than spreading the infection as Mirai. Therefore, Gafgyt data is more similar to benign data than Mirai data. This difference between those attacks can be seen in Fig 4.

We can conclude that SOMKNN is effective for both attack detection and anomaly detection on the datasets. SOM learns the features of different attack types and maps them to the appropriate neurons. KNN improves the accuracy of assigning labels to those neurons.

6. Summary

We have proposed the hybrid of SOM and KNN (SOMKNN) for identifying IoT malware. Our results on the NBaIoT data demonstrate that SOMKNN often exhibits superior performance in comparison to other methods such as SOMSVM, SOMSM, and SOMRF in various experiments. Furthermore, SOMKNN can identify known and unknown attacks on different devices. This is a significant finding as it suggests that our proposed method can be effectively applied to real-time detection in network systems. Additionally, we found that tuning the SOM hyperparameters with different optimization algorithms improves performance. Specifically, we discovered that ATPE is more effective for attack detection and TPE is more effective for unknown attack detection. These findings suggest that our method can be further developed and optimized for better performance.

In future work, we plan to further improve our method by incorporating a new regularized AutoEncoder to preprocess the data before applying SOMKNN. This can improve the accuracy by combining SOMKNN with other methods like AE or PCA in one stage. Additionally, we also plan to use regularizers to enhance data clustering. They have the potential to help separate the data more effectively. Finally, we will test our method on new datasets to validate its effectiveness and confirm its applicability to a wider range of data.

References

[1] M. Abomhara and G. M. Køien, "Cyber security and the internet of things: vulnerabilities, threats, intruders and attacks," *Journal of Cyber Security and Mobility*, pp. 65–88, 2015. doi: 10.1007/978-981-16-7182-1_31

[2] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017. doi: 10.1109/MC.2017.201

[3] S. Zhu, X. Ji, W. Xu, and Y. Gong, "Multi-labelled classification using maximum entropy method," in *SIGIR 2005 - Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2005. doi: 10.1145/1076034.1076082. ISBN 1595930345 pp. 274–281.

[4] M. L. Zhang and Z. H. Zhou, "Multilabel neural networks with applications to functional genomics and text categorization," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 10, pp. 1338–1351, 10 2006. doi: 10.1109/TKDE.2006.162

[5] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990. doi: 10.1109/5.58325

[6] A. Allahdadi, D. Pernes, J. S. Cardoso, and R. Morla, "Hidden Markov models on a self-organizing map for anomaly detection in 802.11 wireless networks," *Neural Computing and Applications*, vol. 33, no. 14, pp. 8777–8794, 2021. doi: 10.1007/s00521-020-05627-7

[7] M. Toshpulatov and N. Zincir-Heywood, "Anomaly Detection on Smart Meters Using Hierarchical Self Organizing Maps," in *2021 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*. IEEE, 2021. doi: 10.1109/CCECE53047.2021.9569097 pp. 1–6.

[8] A. von Birgelen and O. Niggemann, "Anomaly detection and localization for cyber-physical production systems with self-organizing maps," in *Improve-innovative modelling approaches for production systems to raise validatable efficiency*. Springer Vieweg, Berlin, Heidelberg, 2018, pp. 55–71. doi: 10.1007/978-3-662-57805-6_4

[9] N. Li, K. Jiang, Z. Ma, X. Wei, X. Hong, and Y. Gong, "Anomaly Detection Via Self-Organizing Map," in *2021 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2021. doi: 10.48550/arXiv.2107.09903 pp. 974–978.

[10] H. N. Nguyen, V. C. Nguyen, N. N. Tran, and V. L. Cao, “Feature Representation of AutoEncoders for Unsupervised IoT Malware Detection,” in *International Conference on Future Data and Security Engineering*. Springer, 2021. doi: 10.1007/978-3-030-91387-8_18 pp. 272–290.

[11] H. N. Nguyen, N. N. Tran, T. H. Hoang, and V. L. Cao, “Denoising Latent Representation with SOMs for Unsupervised IoT Malware Detection,” *SN Computer Science*, vol. 3, no. 6, pp. 1–15, 2022. doi: 10.1007/s42979-022-01344-1

[12] T. Kohonen, “Essentials of the self-organizing map,” *Neural Networks*, vol. 37, pp. 52–65, 2013. doi: 10.1016/j.neunet.2012.09.018

[13] D. E. Rumelhart and D. Zipser, “Feature discovery by competitive learning,” *Cognitive Science*, vol. 9, no. 1, pp. 75–112, 1 1985. doi: 10.1016/S0364-0213(85)80010-0

[14] S. Sun, Z. Cao, H. Zhu, and J. Zhao, “A survey of optimization methods from a machine learning perspective,” *IEEE transactions on cybernetics*, vol. 50, no. 8, pp. 3668–3681, 2019. doi: 10.1109/TCYB.2019.2950779

[15] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, “Algorithms for hyper-parameter optimization,” *Advances in neural information processing systems*, vol. 24, 2011. doi: https://dl.acm.org/doi/10.5555/2986459.2986743

[16] T. Yu and H. Zhu, “Hyper-parameter optimization: A review of algorithms and applications,” *arXiv preprint arXiv:2003.05689*, 2020. doi: 10.48550/arXiv.2003.05689

[17] G. Luo, “A review of automatic selection methods for machine learning algorithms and hyper-parameter values,” *Network Modeling Analysis in Health Informatics and Bioinformatics*, vol. 5, no. 1, pp. 1–16, 2016. doi: 10.1007/s13721-016-0125-6

[18] J. Bergstra, D. Yamins, and D. Cox, “Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures,” in *International conference on machine learning*. PMLR, 2013. doi: https://dl.acm.org/doi/10.5555/3042817.3042832 pp. 115–123.

[19] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici, “N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders,” *IEEE Pervasive Computing*, vol. 17, no. 3, pp. 12–22, 2018. doi: 10.1109/MPRV.2018.03367731

[20] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, “Kitsune: an ensemble of autoencoders for online network intrusion detection,” *arXiv preprint arXiv:1802.09089*, 2018. doi: 10.48550/arXiv.1802.09089

Manuscript received 20-06-2022; Accepted 12-06-2023.

■

Huu Noi Nguyen received the B.Sc. degree in applied mathematics and informatics from Lipetsk State University, Lipetsk, Russia. He is currently studying the Ph.D. program in Computer Science at Le Quy Don Technical University. His current research interests include Machine Learning, Anomaly Detection, IoT and Information Security.
Email: noi.nguyen@lqdtu.edu.vn

Tuan Hao Hoang graduated from Le Quy Don Technical Unversity in 2001. He received PhD in Computer Science from University of New South Wales, 2009. Currently, he is the Senior Lecturer of Information Security Dept., Institute of Information and Communication Technology, Le Quy Don Technical University. His research interests are related to Artificial Intelligence, Evolutionary computation and Cyber security.
Email: haoth@lqdtu.edu.vn

Van Loi Cao received the B.Sc. and M.Sc. degree in computer science from Le Quy Don Technical University, Hanoi, Vietnam, and the Ph.D degree from University College Dublin, Dublin, Ireland. He is currently the Head of Information Security Department, the Faculty of Information Technology, Le Quy Don Technical Unversity. His current research interests include Deep Learning, Machine Learning, Anomaly Detection, IoT Security, and Information Security.
Email: loi.cao@lqdtu.edu.vn

Nguyen Ngoc Tran is an Associate Professor and the Head of Cyber Security Group at Le Quy Don Technical University in Vietnam. He received PhD in System analysis, control and information processing from Don State Technical University, Russia. His research interests focus on pattern recognition, cyber security and artificial intelligence.
Email: ngoctn@lqtdu.edu.vn

KẾT HỢP BẢN ĐỒ TỰ TỔ CHỨC VỚI K-NEAREST NEIGHBORS CHO PHÁT HIỆN MÃ ĐỘC IOT

Nguyễn Hữu Nội, Hoàng Tuấn Hảo, Cao Văn Lợi, Trần Nguyên Ngọc

Tóm tắt

Trong Internet vạn vật, các thiết bị cảm biến tạo ra lượng dữ liệu khổng lồ trên nhiều lĩnh vực và ứng dụng khác nhau. Các cuộc tấn công do mã độc IoT gây ra cũng ngày càng phổ biến. Để phát hiện phần mềm độc hại IoT, trong nghiên cứu trước đây chúng tôi áp dụng các kỹ thuật phân tích để khám phá thông tin có giá trị cũng như giảm số chiều của dữ liệu gốc. Cụ thể, chúng tôi kết hợp phân loại dựa trên Bản đồ Tự tổ chức (SOM) với AutoEncoders để tạo ra một nền tảng cho phát hiện phần mềm độc hại IoT. Tuy nhiên, thuật toán phân loại dựa trên SOM có một hạn chế là khi các mẫu dữ liệu mới không ánh xạ vào các nơ-ron được gắn nhãn trong bản đồ SOM, khi đó dữ liệu có thể bị phân loại sai. Để giải quyết vấn đề này, trong nghiên cứu này, chúng tôi tiến hành các thí nghiệm dựa trên sự kết hợp của thuật toán phân loại dựa trên SOM với các thuật toán học máy phổ biến như K-Nearest Neighbors, Support Vector Machine, Softmax, Random Forest. Bên cạnh đó, chúng tôi cũng nghiên cứu các phương pháp tối ưu hóa siêu tham số cho SOM. Các phương pháp đề xuất của chúng tôi được kiểm tra trên bộ dữ liệu NBaIoT với các thí nghiệm và cài đặt khác nhau. Từ các kết quả thu được, SOMKNN hoạt động tốt hơn so với SOM đơn thuần cũng như các thuật toán khác.

Từ khóa

SOM, KNN, IoT, tối ưu siêu tham số, phát hiện mã độc.