https://doi.org/10.1016/j.future.2021.02.015

Research on Unsupervised Feature Learning for Android Malware Detection based on Restricted Boltzmann Machines

Zhen Liu[ac], Ruoyu Wang[b], Nathalie Japkowicz[d], Deyu Tang[ac], Wenbin Zhang[e], Jie Zhao[f]

[a]School of Medical Information Engineering, Guangdong Pharmaceutical University, Guangzhou, 510006, China

[b]Information and Network Engineering and Research Center, South China University of Technology, Guangzhou 510041, China

[c]Guangdong province precise medicine and big data engineering technology research center for traditional Chinese medicine, Guangzhou 510006, China

[d]Department of Computer Science, American University, Washington, DC, 20016, USA

[e]University of Maryland, Baltimore County, MD 21250 USA

[f]Department of Information Management Engineering, School of Management, Guangdong University of Technology, Guangzhou, 510520, China

Abstract: Android malware detection has attracted much attention in recent years. Existing methods mainly research on extracting static or dynamic features from mobile apps and build mobile malware detection model by machine learning algorithms. The number of extracted static or dynamic features maybe much high, e.g. thousands of Permissions features in OmniDroid dataset for containing all possible Permissions. As a result, the data suffers from high dimensionality. In addition, to avoid being detected, malware data is varied and hard to obtain in the first place. To detect zeroday malware, unsupervised malware detection methods were applied. In such case, unsupervised feature reduction method is an available choice to reduce the data dimensionality. In this paper, we propose an unsupervised feature learning algorithm called Subspace based Restricted Boltzmann Machines (SRBM) for reducing data dimensionality in malware detection. Multiple subspaces in the original data are firstly searched. And then, an RBM is built on each subspace. All outputs of the hidden layers of the trained RBMs are combined to represent the data in lower dimension. The experimental results on OmniDroid, CIC2019 and CIC2020 datasets show that the features learned by SRBM perform better than the ones learned by other feature reduction methods when the performance is evaluated by clustering evaluation metrics, i.e., *NMI*, *ACC* and *Fscore*.

Keywords: Mobile malware detection, Unsupervised feature learning, Restricted Boltzmann Machines, feature subspaces

1. Introduction

Smart phones have been widely used in people's daily life, such as online banking, automated home control, and entertainment [1]. The number of mobile phone users is expected to rise to 3.8 billion in 2021, marking a 52 percent increase in a relatively short period of five years[2]. Mobile malware is malicious software specifically designed to target mobile devices, such as smartphones and tablets, with the goal of gaining access to private data[3]. Recent reports from AV-Test Institute show that 350,000 new pieces of malware are detected every day[4]. The Kaspersky Lab reports that the number of malicious mobile banker packages circulating online grew by 58% during the first quarter of 2019[5]. G-Data Security expert counted 4.18 million malware applications until the end of the year 2019 and discovered over 7,50,000 new malware applications during the first quarter of 2020[6]. Some recent well-known malware examples

* Corresponding Author: rywang@scut.edu.cn(Ruoyu Wang)

include ransomware, spyware, malicious apps and financial malware[7]. These malicious programs posed serious threats to mobile device users, including stealing user credentials, auto-dialing premium number, and sending SMS message without the user's consent[1]. These threats have made Cybersecurity become one of the main concerns to be addressed by society[8].

In order to protect mobile users from attacks[9], different kinds of techniques have been researched, such as designing privacy aware Apps[10], analyzing the requested Permissions in apps[11][12] and researching malware detection method[13][16]. The anti-virus products, such as Norton, Lookout and Coodo Mobile security, mainly use signature-based matching methods to detect attacks[1]. A malware signature is created by extracting binary patterns, or random snippets, from a sample[13]. Anti-virus companies usually use cryptographic hash, e.g., MD5, to generate a signature for an application[14]. Then any app encountered in the future with the same signature is considered a sample of that malware [13]. However, malware can easily bypass signature-based identification by changing small pieces of its software without affecting the semantics.

A variety of machine learning based techniques have been researched to detect Android malware. These methods include static analysis-based methods and dynamic analysis-based methods. The static analysis-based methods utilize reverse-engineering techniques to decompile the codes of apps from the installing packages. They further extract static features, such as request Permission sequences[15], API (Application Program Interfaces) sequences[16][17], meta-information[18] from the decompiled codes. The detection model is trained on the data characterized by static features. Dynamic analysis-based methods extract reliable information from the execution traces of apps in a controlled environment. The execution traces include system calls[19], traffic traces etc.[7]. The malware detection model is trained on the data characterized by dynamic features extracted from the execution traces.

The number of extracted static or dynamic features may be much high, e.g., the thousands of features in OmniDroid dataset[37] shown in Table 4. As a result, the data may suffer from the high dimensionality problem. That is, the data may include irrelevant and redundant features, and increase the time consumption on training malware detection models. Hence, feature reduction is required as a preprocessing for machine learning based malware detection. The supervised feature reduction methods have been utilized in the field of malware detection. An open problem in this field is that it lacks the researches on unsupervised feature reduction. Clustering (unsupervised machine learning) methods have also been used for malware detection[20], especially when used for detecting zeroday malware[21]. When using clustering for malware detection, unsupervised feature reduction method is an available choice to reduce the data dimensionality. This paper researches the unsupervised feature learning for mobile malware detection, aiming at learning the latent features and reducing the data dimensionality. The contributions of this paper include:

(1)We propose an unsupervised feature reduction method named SRBM(Subspace based Restricted Boltzmann Machines) for mobile malware detection by applying unsupervised feature learning and subspace learning to consider the non-linear relationship among features and the underlying structure of data.
(2)We employ RBMs(Restricted Boltzmann Machines) to learn the reduced feature set by considering the non-linear relationship among features.
(3)We introduce subspace learning into RBMs and search the subspaces by clustering the features in the full feature set.

(4)We evaluate the performance of SRBM by comparing it with RBM, Stacked Auto Encoder (SAE), PCA (Principal Components Analysis) and Agglomeration algorithms in multiple cases. The experimental results on the real datasets show that the *NMI*(normalized mutual information), *ACC*(Accuracy) and *Fscore* are respectively improved about 6.2%, 6.9% and 15.4% on average over all datasets when our method is compared with RBM. And it also outperforms other methods on most datasets in terms of the three metrics.

The rest of this paper is organized as follows. The related works are introduced in Section 2. The RBM and our proposed method are presented in Section 3. The experimental datasets and performance evaluation metrics are described in Section 4. The experimental results are reported in Section 5. Finally, the conclusion of the paper is provided in Section 6.

2. Related work

In this section, we firstly introduce the static, dynamic and hybrid analysis-based methods in mobile malware detection research filed. Then we further overview the related works of feature reduction methods in mobile malware detection research field.

2.1 Mobile malware detection methods

The mobile malware detection methods can be categorized into static analysis-based methods and dynamic analysis-based methods. Both methods have pros and cons. Static analysis is prone to obfuscation but is generally faster and less resource intensive than dynamic analysis. Dynamic analysis is resistant to obfuscation but can be hampered by anti-virtualization and code coverage limitations[22].

Table 1 Static analysis-based methods

Ref.	**Static Features**	**#Features/Feature selection method**	**Methods**
Yen et al. [30]	TF-IDF on code	Not given/None	CNN(Convolutional Neural Network)
Taheri et al.[23]	API, Intent, Permission	Not given/Random Forest Regressor	FNN(First nearest neighbors), ANN(all NN), ANN(weighted ANN), KMNN (k- medoid based NN)
Badhani et al.[24]	API tags, Permissions	217/remove the Features own a constant value or zero variance	Clustering and ensemble methods
Scalas et al.[38]	API packages, classes, methods	41027/information gain	Random Forest
Mercaldo et al.[27]	Images on binary codes	256/None	Deep Neural Networks
Yerima, et al.[22]	Permission, Intents	350/information gain	Fusion method on Random Tree, REPTree, J48 and Voted Perceptron
Pai et al. [21]	Opcode sequences	Not given/None	K-means, Expectation-Maximization
Thiyagarajan et al. [20]	Permissions	130/PCA, Chi, APriori, SPR, PRNR	Decision trees, K-means
Mariconti et al. [29]	Markov chains on API call graph	Not given / None	Random Forest, 1NN,3NN and SVM

In the field of static analysis-based methods, as shown in Table 1, reverse engineering toolkits (such as Apktool) are generally used to decompile codes or to access the different files contained in the installing packages (e.g., the Android manifest.xml and classes.dex). After the decompiling,

we can obtain readable information such as a list of API calls or required Permissions, Intent filters, process names etc. Then, malware app detection rules or detection models based on machine learning could be built from these decompiled data characterized by static features. Different kinds of feature vectors have been extracted, such as binary values of API calls, Intent, Permissions etc. signifies the presence and absence of the feature. [23][24][25]. Some other methods further extract word2vec[26], TF-IDF[7], images[27] of bytecodes, API call graph[28], HIN[1] and Markov chains[29] on the API calls. A variety of machine learning algorithms have been performed on the data with static features, such as CNN[30], FNN[23], Clustering and ensemble[24], Random Forest etc[38]. For example, Pai et al. [21] compute clusters using the well-known K-means and Expectation Maximization algorithms, with the underlying scores based on Hidden Markov Models. Their method obtains 70% to 80% AUC for silhouette coefficient scores with different number of clusters for classifying malware families. On the data with high dimensionality, feature reduction method is used to reduce features, such as information gain[38] [22].

In the field of dynamic analysis-based methods, as shown in Table 2, the methods leverage an emulator or even a physical to run the apps while a monitoring agent captures a series of indicators, such as hardware components accessed, network traffic, system calls invoked, or API calls invoked. Similar to the static analysis methods, different kinds of feature vectors were built on these extracted raw data, such as the n-grams of system calls[31], API call sequences[32][33] and flow statistic features on network traffic[34]. And machine learning methods are used for detecting malwares, such as Autoencoders[35], LSTM[36], K-means and KNN[32].

Table 2 Dynamic analysis-based methods

Ref.	**Dynamic Features**	**#Features/Feature selection methods**	**Methods**
Angelo ett al. [35]	Invoked API call images	(450*450)/None	Autoencoders, SoftMax neural network
Xiao et al. [36]	System call sequences,	Not given/ None	LSTM(Long Short Term Mermory network)
Ananya et al. [31]	n-grams of system calls,	Not given/SAILS	Logistic Regression, CART, Random Forest, XGBoost and Deep Neural Networks
Duarte-Garcia et al. [32]	API calls sequences	Not given/None	K-means and KNN
Shamsi et al. [33]	API calls sequences	Not given/None	Weighted Pair Group Method with Arithmetic Mean

In the field of hybrid analysis-based methods, as shown in Table 3, the static features and dynamic features are used for the training mobile malware detection model. Martin et al.[37] publicly shared OmniDroid dataset. They extracted static features (API, FlowDroid, Permissions, Receivers, Services etc.) and dynamic features based on the Markov chins representation (states sequences and transactions probabilities) of the executed action when app running. They carried out a series of experiments on their datasets. The results show that Random Forest performs the best among AdaBoost, Bagging, ExtraTrees, Gradient Boosting, Random Forest and Voting. Among the different combinations of static feature sets, the API, API+Permission (union of API and Permission) and API+FlowDroid (union of API and Permission) perform much better than other static feature sets. Random Forest with API+ FlowDroid feature set obtains 0.892 accuracy and 0.892 precision. And the union of transactions and frequencies of the dynamic features performs better than one set of them. Random Forest with the combination dynamic features obtains 0.785 accuracy and 0.785 precision. Taheri et al.[39] also publicly shared benchmark

datasets with malware static and dynamic features. The static features include Permissions and Intents. The dynamic features are composed by API calls and network flows. They firstly perform Random Forest for classifying malware and benign on the data with static features, and then perform Random Forest on classifying malware categories on the data with dynamic features.

Table 3 Hybrid analysis-based methods

Ref.	**Hybrid Features**	**#Features/Feature selection methods**	**Methods**
Saif et al. [41]	Permissions, Services, Receivers, Activities, API calls, System calls, etc.	Not given/Relief	Deep belief network
Alzaylaee et al.[40]	Application, Actions/Events,Permission	420/Information gain	Deep Neural Networks
Martin et al. [37]	API, FlowDroid, Permissions, Receivers, Services etc., Markov chains on system calls	2128 API, 961 FlowDroid, 5501 Permissions, 6415 Receivers, 4365 Services, etc., 5932 Markov chains on system calls/None	AdaBoost, Bagging, ExtraTrees, Gradient Boosting, Random Forest
Taheri et al. [39]	Permission, Intents, flow features, API call features	8115 Permission and Intents, 80 network-flow features, 911 API call features/None	Random Forest

The feature vectors extracted from different fields are directly concatenated and are used for training a malware detection model. In this way, the data suffers from the high dimensionality problem. In some papers, supervised feature selection methods, such as information gain and Relief, have been used to reduce the data dimensionality as shown in Tables 1 to 3.

2.2 Feature reduction methods in mobile malware detection

This section further introduces the feature reduction methods in mobile malware detection field. Feizollah et al. [42] overview feature selection methods in mobile malware detection research field, and they claim only 8 out of 100 papers work on feature reduction by feature selection algorithms. Most works select features (choosing API, permission as static features or system calls as dynamic features) based on rationalizing. Feature selection is performed using supervised feature ranking algorithms, such as Information Gain.

Alam et al.[28] take out signatures common in malware and benign instances, to reduce the number of features. Taheri et al.[23] apply Random Forest Regressor algorithm for feature reduction. In addition, they repeat the experiments for {10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, 100%} of the manifest features with higher ranks to determine the optimal features. Badhani et al. [24] remove the irrelevant features from the full feature set. These irrelevant features are identified as such if they have a constant value or zero variance. Mariconti et al. [29] perform PCA (Principal Components Analysis) on Markov chains-based features. Anaya et al.[31] propose a new feature selection mechanism known as selection of relevant attributes for improving locally extracted features using classical feature selectors (SAILS). SAILS is built on top of conventional feature selection methods, such as mutual information, distinguishing feature selector and Galavotti-Sebastiani-Simi. Suresh et al. [43] employ recursive feature elimination (RFE) to explicitly determine the tradeoff between the features used and the accuracy. Lashkari et al. [44] conduct two feature selection algorithms, namely CfsSubsetEval with Best First search method and Information gain with the Ranker search over the training dataset. Sheen et al.[45] compare three feature selection methods (i.e., Chi-Square, Relief and Information Gain) on the dataset characterized by API and Permissions. The experimental results show that the Relief

method performs the best. Thiyagarajan et al. [20] handle the feature reduction on the Permissions features by PCA, Chi-square, APriori, SPR(support-based pruning) and PRNR(permission ranking with negative rate), and then train decision tree model on the data with reduced features; and they further category the malware samples into 45 families using K-means.

Tables 1 to 3 show that most works utilize supervised feature reduction algorithms to reduce the data dimensionality in mobile malware detection. A few papers utilize unsupervised feature extraction method PCA in this field. To the best of our knowledge, this paper is the first work that researches the performance of unsupervised feature learning for mobile malware detection. The unsupervised feature learning method could be used in the case of clustering-based malware detection cases such as detecting zeroday malware.

3. Unsupervised feature learning method

This section firstly introduces existing RBM method and then presents our SRBM method.

3.1 RBM

RBMs have been used effectively in modeling distributions [46]. An RBM[47] is an undirected graphical model. It is shown in Fig.1. It consists of visible variables $\mathbf{v} \in \{0,1\}^{n_v}$ to represent observable data and stochastic hidden variables $\mathbf{h} \in \{0,1\}^{n_h}$ to capture dependencies between observed variables[48]. Each visible variable is connected to each hidden variable. And there is no connection in the same visible or hidden layer. After the learning process, the output of the hidden layer is the learned latent feature vector and could be used as the input to the machine learning algorithms for training the malware detection model. The goal of RBM is to find $P_{data}(\mathbf{v})$ the unknown true high dimensional distribution of the visual layer variables. It means that the features could be learned without labels. It is an unsupervised feature learning method.

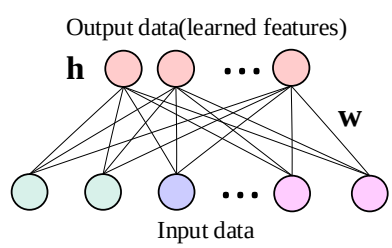


Fig.1 RBM

The energy function of RBM is defined as

$$E(\mathbf{v},\mathbf{h}) = -\sum_{i=1}^{n_v}\sum_{j=1}^{n_h} v_i W_{ij} h_j - \sum_{i=1}^{n_v} a_i v_i - \sum_{j=1}^{n_h} b_j h_j \tag{1}$$

Where W_{ij} denotes a real valued weight between visible unit v_i and hidden unit h_j; a_i and b_j are real valued bias terms associated with the ith visible unit and the jth hidden unit respectively; n_v and n_h are the number of visual units and the number of hidden units respectively. The joint distribution of $\mathbf{v}$ and $\mathbf{h}$ is defined as

$$P(\mathbf{v},\mathbf{h}) = \frac{1}{Z}\exp(-E(\mathbf{v},\mathbf{h})) \tag{2}$$

Where Z is the normalizing constant. It is defined as

$$Z=\sum_{\mathbf{v}}\sum_{\mathbf{h}}\exp(-E(\mathbf{v},\mathbf{h})) \tag{3}$$

The activation state of each hidden layer unit is conditionally independent for a given state of the visible layer unit. The conditional distributions over hidden **h** and visible **v** vectors are derived from Eqs.(1) and (2) as

$$P(\mathbf{h}\,|\,\mathbf{v})=\prod_{j=1}^{n_h}P(h_j\,|\,\mathbf{v}),\text{with } P(h_j=1|\,\mathbf{v})=g(\sum_{i=1}^{n_v}W_{ij}v_i+b_j) \tag{4}$$

$$P(\mathbf{v}\,|\,\mathbf{h})=\prod_{i=1}^{n_v}P(v_i\,|\,\mathbf{h}),\text{with } P(v_i=1|\,\mathbf{h})=g(\sum_{j=1}^{n_h}W_{ij}h_j+a_i) \tag{5}$$

Where g denotes the sigmoid activation function $g(x)=1/(1+e^{-x})$. We assume that a training set is denoted by $S=\{\mathbf{v}^1,\mathbf{v}^2,...,\mathbf{v}^{n_s}\}$, $\mathbf{v}^i=(v_1^i,v_2^i,...,v_{n_v}^i)^T$, i=1,2,..,n_s and these samples are independent and identically distributed. The $\mathbf{v}^i$ denotes the ith sample in S, and there are n_v features to represent a sample $\mathbf{v}^i$, and the samples size of S is n_s. The object of RBM is to maximize the log-likelihood of $P(\mathbf{v})$ that is defined as

$$L=\ln\prod_{m=1}^{n_s}P(\mathbf{v}^m)=\sum_{m=1}^{n_s}\ln P(\mathbf{v}^m) \tag{6}$$

$$P(\mathbf{v}^m)=\frac{1}{Z}\sum_{\mathbf{h}}\exp(-E(\mathbf{v}^m,\mathbf{h})) \tag{7}$$

$$\begin{aligned}L&=\sum_{m=1}^{n_s}(\ln\sum_{\mathbf{h}}\exp(-E(\mathbf{v}^m,\mathbf{h}))-\ln(\sum_{\mathbf{v},\mathbf{h}}\exp(-E(\mathbf{v},\mathbf{h}))))\\&=\sum_{m=1}^{n_s}(\ln\sum_{\mathbf{h}}\exp(-E(\mathbf{v}^m,\mathbf{h})))-n_s\ln(\sum_{\mathbf{v},\mathbf{h}}\exp(-E(\mathbf{v},\mathbf{h})))\end{aligned} \tag{8}$$

These parameters are updated using the gradient ascent method. The partial deviation (i.e., the gradient) of L with respect to the parameters $\{\mathbf{W},\mathbf{a},\mathbf{b}\}$ can be obtained as below.

$$\frac{\partial L}{\partial W_{ij}}=\sum_{m=1}^{n_s}[P(h_j=1|\,\mathbf{v}^m)v_i^m-\sum_{\mathbf{v}}P(\mathbf{v})P(h_j=1|\,\mathbf{v})v_i] \tag{9}$$

$$\frac{\partial L}{\partial a_i}=\sum_{m=1}^{n_s}[v_i^m-\sum_{\mathbf{v}}P(\mathbf{v})v_i\,] \tag{10}$$

$$\frac{\partial L}{\partial b_j}=\sum_{m=1}^{n_s}[P(h_j=1|\,\mathbf{v}^m)-\sum_{\mathbf{v}}P(\mathbf{v})P(h_j=1|\,\mathbf{v})] \tag{11}$$

To avoid the exponential complexity of summing over all values of the visible variables when calculating the second terms of (9), (10), and (11), one can approximate this expectation by the samples from the model distribution. These samples can be obtained by Gibbs sampling(CD-k algorithm[49]). The Gibbs chain is initialized with a training example $\mathbf{v}^{(0)}$ of the training set and yields the sample $\mathbf{v}^{(k)}$ after k steps of Gibbs sampling. Then the Eqs.(9),(10),(11) are estimated as (12),(13) and (14) respectively. The $\mathbf{v}^{m(0)}$ denotes the $\mathbf{v}^m$ in the 0th step of Gibbs sampling (i.e., $\mathbf{v}^m$), and $\mathbf{v}^{m(k)}$ denotes the sampled **v** after kth step of Gibbs sampling.

$$\frac{\partial L}{\partial W_{ij}} = \sum_{m=1}^{n_s} [P(h_j = 1 | \mathbf{v}^{m(0)}) v_i^{m(0)} - P(h_j = 1 | \mathbf{v}^{m(k)}) v_i^{m(k)}] \tag{12}$$

$$\frac{\partial L}{\partial a_i} = \sum_{m=1}^{n_s} [v_i^{m(0)} - v_i^{m(k)}] \tag{13}$$

$$\frac{\partial L}{\partial b_j} = \sum_{m=1}^{n_s} [P(h_j = 1 | \mathbf{v}^{m(0)}) - P(h_j = 1 | \mathbf{v}^{m(k)})] \tag{14}$$

The CD-k algorithm for updating the gradient approximation of the parameters is shown as Algorithm 1[48]. The RBM(**W, a, b**) denotes the RBM network with the parameters of **W**, **a** and **b**. k is the number of steps of CD-k algorithm and S_b is the training batch. In lines 1 to 7, it firstly calculates the activation probability of all hidden layer units according to Eq.(4) and then samples hidden units $\mathbf{h}^{(t)}$ from $P(\mathbf{h}|\mathbf{v}^{(t)})$ by Gibbs sampling. Similarly, it calculates the activation probability of all visual layer units according to Eq.(5) and then samples visual units $\mathbf{v}^{(t+1)}$ from $P(\mathbf{v}|\mathbf{h}^{(t)})$ by Gibbs sampling. In lines 8 to 13, the gradient approximation of each parameter is updated according to Eqs. (12) to (14) respectively. The output of Algorithm 1 is the updated gradient approximation of each parameter.

Algorithm 1 k-step contrastive divergence

Input: k=1, training batch S_b, RBM(**W**, **a**, **b**)

Output: Δ**W**, Δ**a**, Δ**b**

1 Δ**W**=0, Δ**a**=0, Δ**b**=0

2 for all the **v** in S_b do

3 $\mathbf{v}^{(0)} \leftarrow \mathbf{v}$

4 for t=0,…,k-1 do

5 for j=1,…,n_h do sample $h_j^{(t)} \sim P(h_j|\mathbf{v}^{(t)})$

6 for i=1,…,n_v do sample $v_i^{(t+1)} \sim P(v_i|\mathbf{h}^{(t)})$

7 end for

8 for j=1,…, n_h, i=1,…, n_v do

9 $\Delta W_{ij}=\Delta W_{ij}+ (P(h_j=1|\mathbf{v}^{(0)})v_i^{(0)}- P(h_j=1|\mathbf{v}^{(k)})v_i^{(k)})$

10 $\Delta a_i=\Delta a_i+ (v_i^{(0)} - v_i^{(k)})$

11 $\Delta b_j=\Delta b_j+(P(h_j=1|\mathbf{v}^{(0)})- P(h_j=1|\mathbf{v}^{(k)}))$

12 end for

13 end for

The training process of RBM is shown in Algorithm 2[48]. The CDK(k,S,RBM(**W**,**a**,**b**)) is obtained by Algorithm 1. J epochs are executed. In line 2, it obtains the gradient approximation of each parameter in each epoch through Algorithm 1. In lines 3 to 6, it updates the parameters by gradient ascent method for maximizing the log-likelihood of $P(\mathbf{v})$. The output of Algorithm 2 (i.e.,

the solved **W**, **a** and **b)** includes the parameters of the trained RBM model that can be used to obtain the new data with learned features, i.e. **X'**=**WX**+**b.**

Algorithm 2 training algorithm of RBM

Input: Training set S, minibatch size n_{block}, epoch: J, learning rate η and k in CD-k, initialize parameters **a**, **b** and **W**.

Output: Trained RBM model

1 for $iter$=1,2,…,J do

2 {Δ**W**, Δ**a**, Δ**b**}=CDK(k,S,RBM(**W**,**a**,**b**))

3 **W**=**W**+ η (Δ**W**/n_{block})

4 **a**=**a**+ η (Δ**a**/n_{block})

5 **b**=**b**+ η (Δ**b**/n_{block})

6 end for

3.2 Subspace based RBMs

In RBMs, the time complexity is related with the number of parameters required to be solved. In addition, these parameters are correlated with the number of input features (i.e., the number of visual units). For the data with thousands of features, there would be tens of thousands of parameters with the object of reducing the data dimensionality into 10 to 100. In addition, to improve the performance, existing unsupervised feature selection methods tend to estimate the underlying structure of the data in the original feature space. Once the structured is found , the subspaces are first searched, and then the feature reduction method is performed on each subspace [50][51]. In this way, the underlying structure of the data could be learned and the number of features in a subspace is smaller than that in the full feature set. So that, the number of parameters of a feature learning model trained on a subspace is much less than the one trained on the data represented by the full feature space. Therefore, we introduce subspaces into RBMs, and propose Subspace based RBMs(SRBM) method. We expect SRBM could improve the performance and decrease the resource consumption of RBMs.

Existing work [52] argued that the use of a large number of random subspaces can significantly benefit the unsupervised feature selection accuracy. However, the large number of random subspaces may increase the time consumption. To decrease the time consumption, this paper applies the clustering algorithm on the full feature set for finding the subspaces. The flowchart of SRBM is shown in Fig.2. On the training data in size of [n_s, n_v], the subspaces are found by dividing full feature set into feature clusters. The T_SubF_i denotes the ith dataset represented by the ith subspace, where i=1,2,...,K. The size of T_SubF_i matrix is [n_s,$n_v^{[i]}$], and n_v= $n_v^{[1]}$ +…+ $n_v^{[K]}$. Then, an RBM is individually trained on each dataset. The RBM_1 to RBM_K are the RBM models trained on $T\_\ SubF_1$ to T_SubF_K respectively. The outputs of RBMs on these K datasets are denoted by T_dSubF_1, T_dSubF_2,…,T_dSubF_K. These are the datasets with reduced feature set obtained by RBMs. The size of T_dSubF_i is [n_s,$n_h^{[i]}$]. All learned features are combined for characterizing the dataset. *T_FinalFeatureSet* denotes the output of our method, and the size of *T_FinalFeatureSet* is [n_v, n_h], where n_h= $n_h^{[1]}$ +…+ $n_h^{[K]}$.

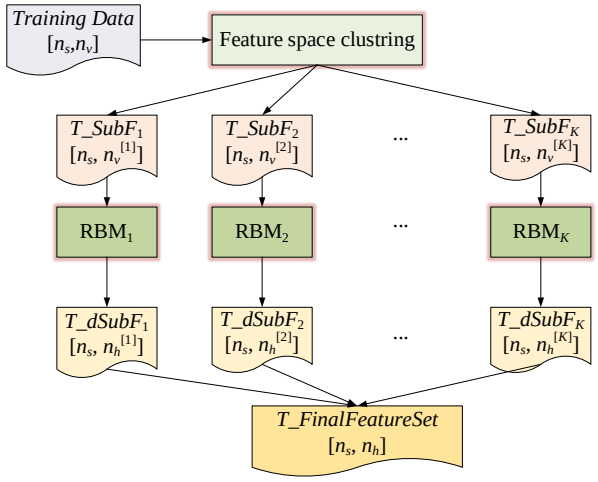


Fig.2 Flow Chart of SRBM

The algorithm of SRBM for unsupervised feature learning is shown in Algorithm 3. It includes model training and data transforming.

In the lines 1 and 2 of model training part, the K subspaces are obtained using K-means on the full feature set. The S.T (in the size of [n_v, n_s]) is the transposition of S. So that Similar features are clustered into a subspace by performing K-means. The **km** is the K-means model for finding feature subspaces. The *SubSet* is a dataset array and each dataset in it denotes the dataset represented by a subspace. In lines 3 to 6, an RBM model is individually trained on the dataset represented by each subspace through Algorithm 2. The *SubSet*[i].T (in the size of [n_s, $n^{[i]}_v$]) is the transposition of *SubSet*[i]. So that RBM learns features on the dataset represented by ith subspace. The *rbmArray* contains all trained RMB models on all feature subspaces. The output of Model Training part includes *rbmArray* and **km**.

In the line 1 of data transforming part, the feature subspaces of the test dataset are firstly acquired by the trained **km**. In lines 2 to 6, the RBM model in *rbmArray* is performed on corresponding feature subspace so as to reduce the dimensionality of the dataset represented by each subspace. After all iterations, all learned features by the RBM models are combined to represent the origin dataset in lower dimension. The *Rdata* denotes the dataset with reduced feature set.

Algorithm 3 SRBM algorithm

Model Training

Input: Training set S, the number of subspaces K

Output: Learned RBM set and trained K-means model

1 **km**=Kmeans(S.T, K) *#performing clustering algorithm on the full feature set*

2 *SubSet*=**km**(S.T)

3 for i=0 to K-1 do

4 model = rbm.train(*SubSet*[i].T)

5 *rbmArray*.append(model)

6 end for

7 return *rbmArray*, **km**

Feature reducing

Input: data set *DS*, learned RBM model set *rbmArray*, learned K-means model km

Output: data with reduced feature

```
SubSet = km(DS.T)
Rdata=[]
for i=0 to len(SubSet)-1 do
  RSet= rbmArray[i].Transform(SubSet[i].T)
  Rdata =Combine(Rdata, RSet)          #combine the features in Rdata and RSet
end for
return Rdata
```

3.3 Analysis on the number of learned parameters

This section analyzes the number of learned parameters by RBM and SRBM. Using SRBM, the number of parameters in RBM could be decreased, so as to decrease the time consumption when training the RBM models. In RBM, the number of parameters is $N_p = n_v + n_v*n_h+n_h$. The n_v is the number of input features, i.e., the number of visual units. The n_h is the number of hidden units, i.e., the number of features that will be leaned. In SRBM, the origin feature space is clustered into multiple subspaces. The number of parameters in SRBM is calculated as Eq. (15), where K is the number of subspaces. The number of hidden units in the 1*st* to the (K-1)*th* subspace is all set as $\lfloor n_h/K \rfloor$, and that in the Kth Subspace is set as n_h-(K-1)* $\lfloor n_h/K \rfloor$. The $N_p^{(s)}$ is smaller than the N_p. The time consumption of SRBM and RBM on the real dataset will be further discussed in Section 5.4.3.

$$\begin{aligned} N_p^{(s)} &= \sum_{i=1}^{K} (n_v^{[i]} + n_v^{[i]} * n_h^{[i]} + n_h^{[i]}) \\ &= \sum_{i=1}^{K} (n_v^{[i]} + n_v^{[i]} * n_h / K + n_h / K) \\ &= n_v + n_v * n_h / K + n_h \end{aligned} \tag{15}$$

4.Datasets and performance evaluation metrics

This section mainly introduces the datasets and performance evaluation metrics used in the following experiments.

4.1 Datasets

The OmniDroid, CIC2019 and CIC2020 datasets are used in the following experiments and they are introduced as below. The number of features and samples in each dataset could be found in Table 4.

(1) OmniDroid dataset

The OmniDroid dataset was publicly shared recently[37]. It includes 11,000 malware and 11,000 benignware samples. The samples are from two different sources, Koodous and AndroZoo respectively. There are 18785 malware samples from Androzoo, 4386 malware samples from Koodous, and 13619 benignware from Koodous. The preprocessing of this dataset done in [37] is as following. Three filtering processes were firstly applied on these samples: removing apks with same package name, removing invalid apks at the aid of AndroGuard tool and removing apks without DroidBox analysis (discarding samples that could not be actually executed in the Android emulator used by DroidBox). The detail of data collection and preprocessing could be further found in [37]. The static features and dynamic features were extracted from this dataset.

The static features include API calls, Opcodes, Package, Permissions, Intent receivers, Intent services, Intent activities, System commands and FlowDroid. FlowDroid features[37] denote the paths to the results obtained by FlowDroid that is defined as the first fully context, field, object and flow sensitive taint analysis which considers the Android application lifecycle and UI widgets, and which features a novel, particularly precise variant of an on-demand alias analysis[53]. The main goal of this set of static features is to provide an insight of the

application's expected behavior and the range of actions that it could take based on a static analysis of the code which does not imply code execution[37]. These features could be obtained before app running.

Different classification algorithms (AdaBoost, Bagging, ExtraTrees, Gradient Boosting, Random Forest and Voting) have been performed on OmniDroid datasets represented by different unions of static feature sets in [37]. The accuracy and precision of detecting malware show that the API, API+Permission (union of API and Permission) and API+FlowDroid (union of API and FlowDroid) perform better than other static feature sets. Therefore, the API, API+Permission and API+FlowDroid feature sets will be used in our following experiments. And the datasets with the three feature sets are respectively named as API, APIPermission and APIFlowDroid. The #feature column in Table 4 shows that the features in API, APIPermision, APIFlowDroid are more than one thousand. Feature reduction is required to decrease the time consumption on training malware detection models.

In contrast to static analysis, the use of a dynamic analysis tool allows to model the real behavior exhibited in a simulated environment where the application is executed[37]. The sequence of actions performed throughout the execution is extracted. Each action is linked to a category (i.e. file access), a timestamp and a series of parameters (i.e. the path of the file accessed). In order to build a feature vector, a Markov chains-based representation was employed. Each state is represented by the category of the action and a series of arguments. The transitions probabilities and frequency of each state were extracted as features. The dataset with these dynamic features is named Dynamic in the following experiments. It owns 5932 features.

Table 4 The number of features and samples in each dataset

Datasets		**#Features**	**#Samples**
OmniDroid	API	2128	11000Benign/11000Malware
	APIPermission	7629	11000Benign/11000Malware
	APIFlowDroid	3089	11000Benign/11000Malware
	Dynamic	5932	11000Benign/11000Malware
CIC2019		8115	1187Benign /407Malware
CIC2020		9503	39931Benign/40923Malware

4.1.2 CIC2019 and CIC2020 datasets

The publicly shared datasets named CICInvesAndMal2019 (CIC2019 in short) [39] and CCCS-CIC-AndMal-2020(CIC2020 in short) [54][55] are also used as our experimental datasets. The CIC2019 dataset [39] includes the requested Permissions and Intent actions as static features which are extracted from ManifestFile.xml of app's Apk files. The 8115 Permission and Intent features were extracted. For each Apk, the appearance number of each extracted Permission or Intent feature is counted as the feature value. There are 407 malware samples and 1187 benign samples. The collected dataset includes 42 malware families, which belong to the following four categories: adware, ransomware, scareware and SMS malware.

To evaluate the performance of unsupervised feature learning methods in the case of zeroday malware detection, CIC2020 dataset[54][55] is also used because this dataset includes zeroday malware. It includes 14 malware categories including adware, backdoor, file infector, no category, Potentially Unwanted Apps (PUA), ransomware, riskware, scareware, trojan, trojan-banker, trojan-dropper, trojan-sms, trojan-spy and zero-day. The main extracted features include activities, Metadata, the Permissions requested by application, system features (such as camera and internet). In total, there are 9503 features in this dataset. The training data includes 32084 benign samples

and 27596 malware samples; the testing data includes 7847 benign samples and 13327 zeroday malware samples in this dataset.

4.2 Performance evaluation metrics

This section introduces the performance evaluation metrics used in the following experiments. To evaluate the performance of the unsupervised feature learning algorithms, the clustering accuracy (*ACC*) and normalized mutual information (*NMI*) are generally used as clustering evaluation metrics[56]. The *ACC* is defined as

$$ACC = \frac{\sum_{i=1}^{n} \delta(p_i, map(q_i))}{n} \tag{16}$$

Where p_i denotes the true labels of the dataset, q_i the clustering labels obtained by the K-means clustering algorithm, and map(q_i) the mapping function that matches the obtained clustering label q_i to the equivalent label of the dataset. The delta function $\delta(a,b)$=1 if a=b, otherwise, $\delta(a,b)$=0.

The *NMI* is the normalized mutual information between the true and predicted labels. It is defined as

$$NMI(P,Q) = \frac{I(P;Q)}{\sqrt{H(P)H(Q)}} \tag{17}$$

Where P denotes the true labels, Q the clustering results, $I(P;Q)$ the mutual information between P and Q, and $H(P)$ and $H(Q)$ the entropies of P and Q respectively.

However, in the field of anomaly detection, the abnormal data are always much less well represented than the normal data, the *ACC* metric may, therefore, be biased towards the normal data. If there are 99 normal samples and 1 abnormal sample in the testing data, the *ACC* could be close to 100%, even when the accuracy of abnormal data is 0%. Based on the p_i and map(q_i), we can also obtain other evaluation metrics such as the *Fscore* of anomaly data calculated on the clustering results. It is defined as

$$\text{Fscore} = \frac{2*\left(\sum_{i=1,p_i\in C_a}^{n} \delta(p_i, map(q_i)) \Big/ n_a\right)*\left(\sum_{i=1,q_i\in C_a}^{n} \delta(p_i, map(q_i)) \Big/ m_a\right)}{\left(\sum_{i=1,p_i\in C_a}^{n} \delta(p_i, map(q_i)) \Big/ n_a\right)+\left(\sum_{i=1,q_i\in C_a}^{n} \delta(p_i, map(q_i)) \Big/ m_a\right)} \tag{18}$$

Where the C_a denotes the normal class, n_a denotes the number of samples in C_a, and m_a denotes the number of samples predicted as C_a by clustering algorithm. $p_i \in C_a$ represents the samples whose true label is C_a, and $q_i \in C_a$ represents the samples whose prediction label is C_a.

The higher the *ACC*, *Fscore* and *NMI* are, the better the performance of the unsupervised feature learning algorithm is.

In some related papers, the classification evaluation metrics are also calculated to evaluate the performance of unsupervised feature reduction algorithm[57]. The *OA* and *F-measure* of anomaly data are used as evaluation metrics used for classification case. The *OA* is defined as

$$OA = \frac{TP + TN}{n} \tag{19}$$

The *F-measure* is the composite evaluation of *recall* (*R*) and *precision* (*P*). If *recall* is improved but *precision* drops significantly, and the *F-measure* could not be improved.

$$F-measure=\frac{2RP}{R+P} \tag{20}$$

$$\text{where } R=\frac{TP}{TP+FN} \qquad P=\frac{TP}{TP+FP}$$

In the above equations, *TP* denotes the True Positives, that is the number of correctly identified abnormal samples; *TN* denotes the True Negatives, that is the number of correctly identified normal samples, *FP* denotes the False Positives, that is the wrongly identified normal samples; and *FN* denotes the False Negatives, that is the number of wrongly identified abnormal samples.

5. Experiments

This section firstly introduces the experiments design and then analyze our experimental results from different aspects.

5.1 Experiments design

In this paper, we mainly carry out experiments from the following five aspects: (1) analysis on the performance in the case of zeroday malware detection, (2) analysis on the performance evaluated by clustering evaluation metrics, (3) analysis on the performance evaluated by classification evaluation metrics, (4) discussion on the parameters of SRBM and (5) discussion on the time consumption of feature reduction methods. In the first three aspects of experiments, we firstly analyze the performance of feature learning algorithms with different number of learned features, and then compare the performance of different feature reduction methods with the optimal number of reduced features.

To evaluate the performance of SRBM, it is compared with RBM, SAE, PCA and Agglomeration. SAE and RBM are the unsupervised feature learning algorithms. PCA and Agglomeration are unsupervised feature extracting methods. PCA has been used in the field of malware detection[20]. Agglomeration also utilizes feature clusters to find the underlying structure of data. RBM is introduced in Section 3.1. The SAE, PCA and Agglomeration are briefly described as below.

(1) SAE

An auto-encoder is an unsupervised back-propagation neural networks algorithm for feature extraction[58]. It aims at minimizing the reconstruction error between input and output. It is a symmetrical structure of artificial neural networks. A single auto-encoder consists of two stages: encoder and decoder. An encoder aims to map an input vector *x* into a hidden representation *h* through an encoding function:

$$h=f(x)=\gamma_1(W_1x+b_1) \tag{21}$$

where γ_1 is a non-linear activation function. The hidden representation *h* is then reconstructed using decoding function in order to generate the output vector *y*. The decoding function γ_2 is also a non-linear mapping function shown as:

$$y=g(h)=\gamma_2(W_2h+b_2) \tag{22}$$

W_1 and W_2 represent the weight matrices of the hidden layer and output layer, respectively. The b_1 and b_2 are the bias vectors of the hidden layer and output layer, respectively.

The object of AEs is to minimize the reconstruction error($\mathcal{L}(x,y)$), which measures the differences between origin input and the consequent reconstruction. The mean square error (MSE) is used as the objective function. The AdaGrad is used to optimize the weights and biases in this paper. The stacked auto-encoder (SAE) [59] is shown as shown in Fig.3. The first layer takes the original data vector x as input to train the parameters of the first hidden layer. The output of the first hidden layer is fed into the second hidden layer to train the parameters. These steps are repeated until the parameters of all hidden layers are trained.

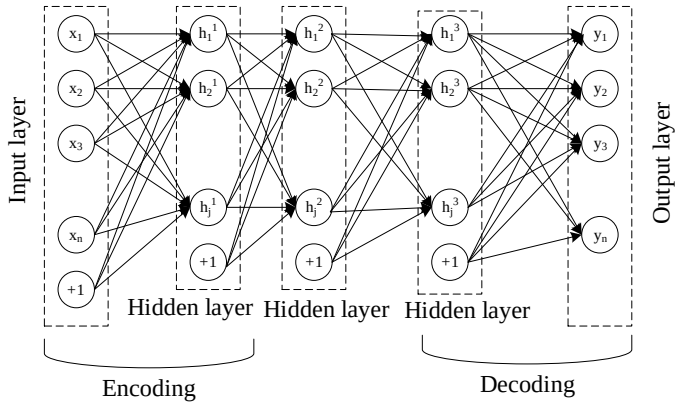


Fig.3 Stacked auto-encoder diagram

(2)PCA

PCA is an unsupervised technique for extracting variance structure from high dimensional datasets[60]. It calculates the eigenvectors from the covariance matrix of the training set, keeping only the first k eigenvectors that correspond to the highest eigenvalues. These k eigenvectors define the feature space, i.e., the reduce feature set. It is an orthogonal projection or transformation of the data into a subspace so that the variance of the projected data is maximized[60].

(3) Agglomeration

Similar to agglomerative clustering, but agglomeration recursively merges features instead of samples[61]. It firstly clusters the features by hierarchical clustering algorithm. And then, in each feature cluster, feature values are average and transferred into a feature.

In the following experiments, the K-means algorithm is used for finding the subspaces in SRBM and the number of clusters of K-means (i.e., the number of subspaces) is set as 10. This parameter will be further discussed in section 5.4.2. The 10-fold cross validation is executed on OmniDroid and CIC2019 datasets. That is, the parameters of the unsupervised learning model are trained on 90% of the data (used as training set), and the model is used to transfer the 10% data (used as testing set) into the reduced dimension data. On the CIC2020 dataset, the training data is used for training the feature learning model and malware detection model, and the models are evaluated on the testing data. The training and testing data in CIC2020 are illustrated in Section 4.1.2. All algorithms are developed using the Python language. Agg and PCA are implemented by invoking the packages from scikit-learn[62].

5.2 Performance evaluated by clustering evaluation metrics

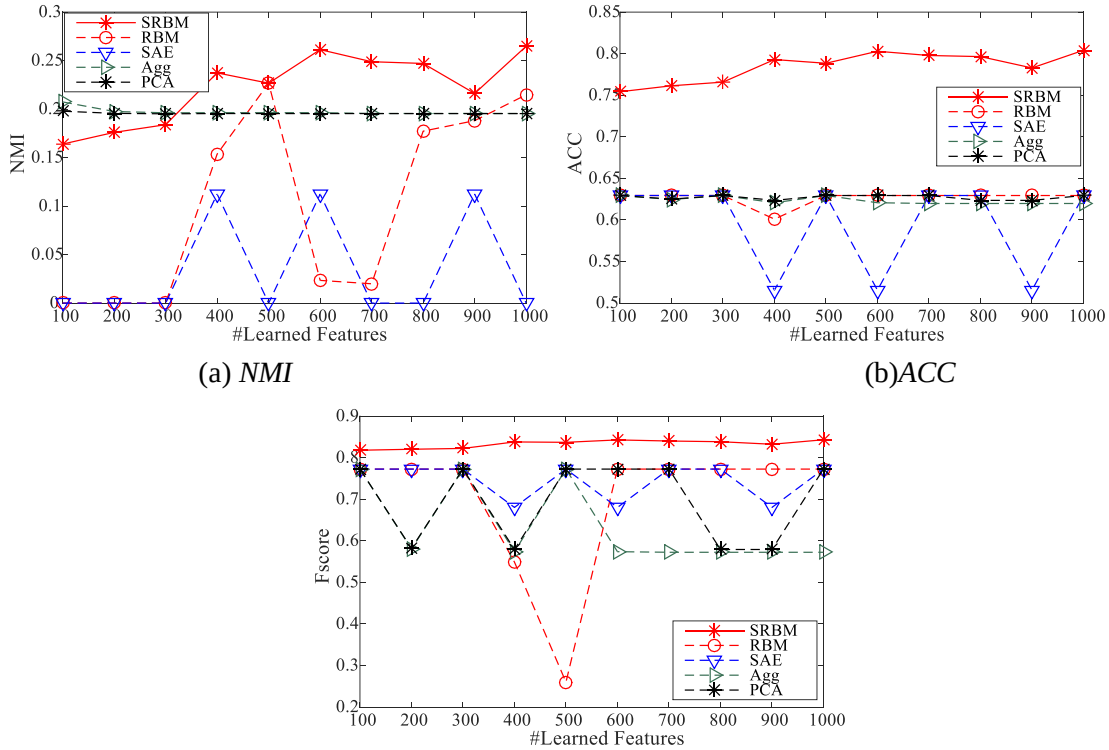
This section reports the experimental results evaluated by clustering evaluation metrics

5.2.1 Performance with different number of features

In this section, we analyze the experimental results of SRBM method against the other methods with different number of learned/extracted features. The number of learned/extracted features is set in the range of {100,200,300, 400,500,600,700,800,900,1000}.

(1) The performance on detecting zeroday malware

In the zeroday malware detection case, clustering algorithms are generally used. In such case, unsupervised feature learning/extracting methods are used for reducing data dimensionality. This is also an application case of our method. This section carries out experiments on the CIC2020 data to evaluate our method in the case of zeroday malware detection. That is, the malware categories in the testing data are not included in the training data. The results are shown in Fig.4. It shows that, SRBM significantly improves the *NMI*, *ACC* and *Fscore* of RBM. In detail, *NMI*, *ACC* and *Fscore* are respectively improved about 0.12, 0.16 and 0.13. When compared with SAE, Agg and PCA, the *ACC* and *Fscore* of SRBM are much better than others in all cases of learned features. In terms of *NMI*, SRBM performs the best when the learned features are more than 400.



(a) *NMI* (b)*ACC*

(d) *Fscore*

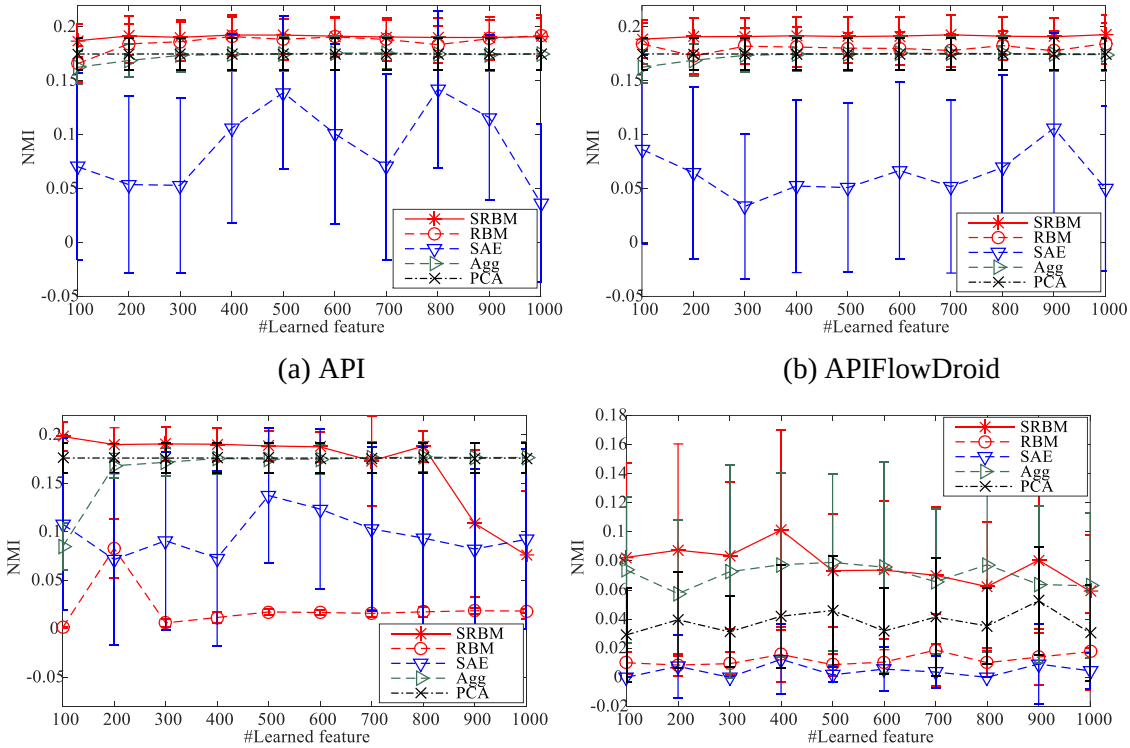
Fig. 4 *NMI*, *ACC* and *Fscore* results in the case of zeroday malware detection

(2) Performance evaluated on the data with static features

The *NMI*, *ACC* and *Fscore* of different unsupervised feature learning/extracting methods are shown in Figs. 5, 6 and 7 respectively. With different numbers of learned/extracted features, there is no trend suggesting that the performance would be better when more features are learned/ extracted.

In terms of *NMI*, Fig.5 shows that SRBM perform the best in most cases. The *NMIs* of Agg and PCA are close to each other. On the API and APIFlowDroid datasets, the SRBM and RBM

perform better than other methods. In addition, SRBM always outperforms RBM and the improvement is significant on APIPermission and CIC2019 datasets. And it also outperforms RBM on API and APIFlowDroid datasets. This illustrates that the SRBM is better than the origin one. This is because that the SRBM benefits from learning the features in the subspace and knows more about the data structure.



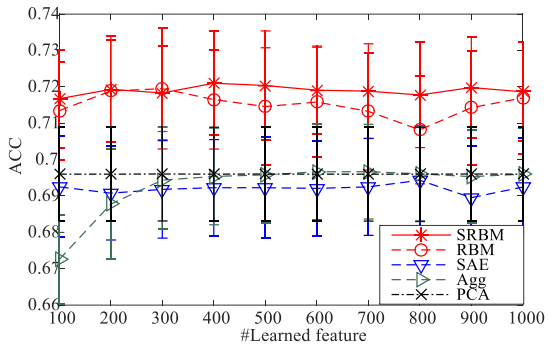
(a) API (b) APIFlowDroid

(c) APIPermission (d) CIC2019

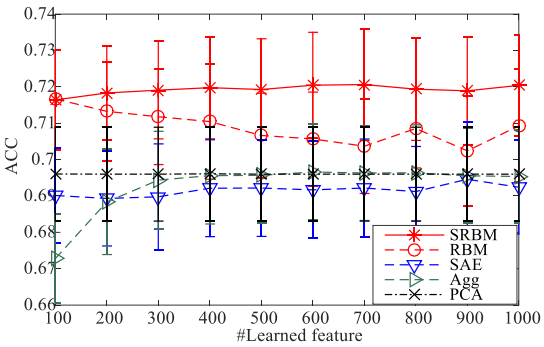
Fig. 5 *NMI* Results of different feature learning/extracting algorithms

In terms of *ACC*, the results are similar to those evaluated by *NMI*. Fig. 6 also shows that SRBM is able to improve the performance of RBM in *ACC*. And it performs better than other methods on all datasets except the *ACC* on APIPermission when learned features are more than 800. The *ACC* is evaluated on benign and malware classes. However, this metric may be biases towards the majority class (the class represented by a large number of samples, i.e. the benign class), and the performance of the minority class (the class represented by a small number of samples i.e. the malware class) may be ignored. For example, there are 407 Malware samples and 1187 benign samples CIC2019 dataset. Therefore, this paper also evaluates the performance in terms of *Fscore* of malware class that is shown in Fig.7. On CIC2019 dataset, the number of malware samples is much smaller than that of benign ones, so the domain is said to suffer from class imbalance problem. The *Fscores* on the CIC2019 dataset are much worse than those on other datasets, while the *ACC* on the CIC2019 dataset is close to those obtained on the other datasets as shown in Fig.6. This is the class imbalance problem. Nevertheless, the results show that SRBM still obtains the best *Fscore* on all datasets. And it improves the *Fscore* of RBM significantly on APIPermission and CIC2019 datasets.

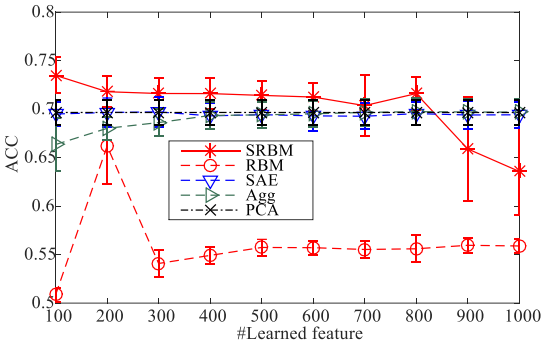
The above results demonstrate that SRBM is able to improve the performance of RBM and outperforms others when they are evaluated by clustering evaluation metrics.



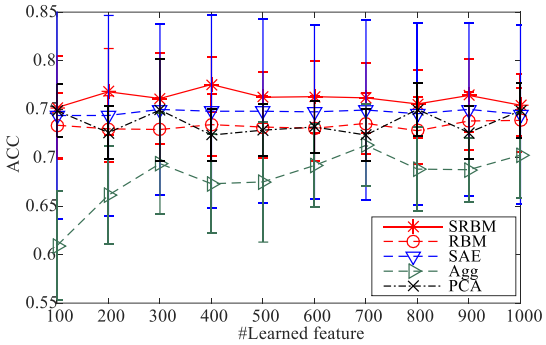
(a) API



(b) APIFlowdroid

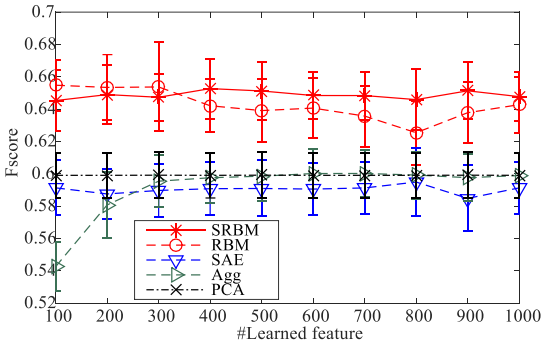


(c) APIPermission

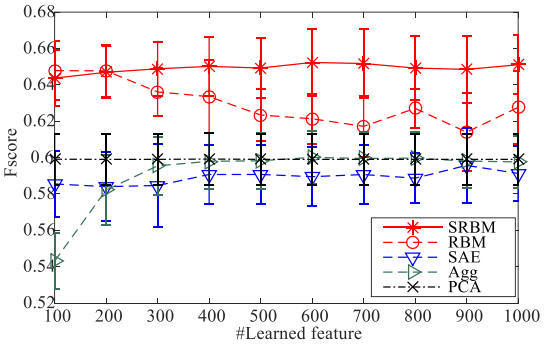


(d) CIC2019

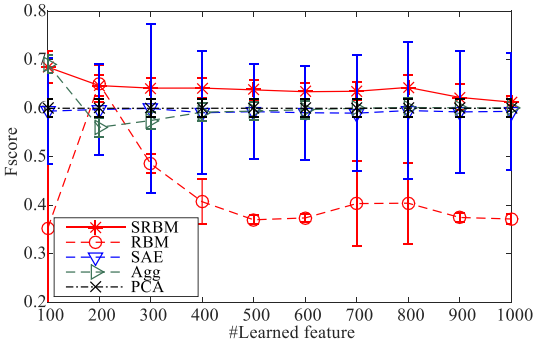
Fig.6 *ACC* Results of different feature learning/extracting algorithms



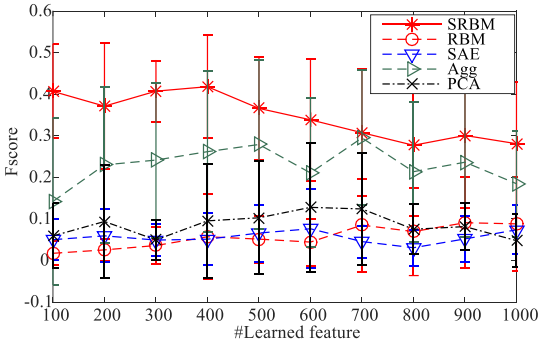
(a) API



(b)APIFlowDroid



(c) APIPermission

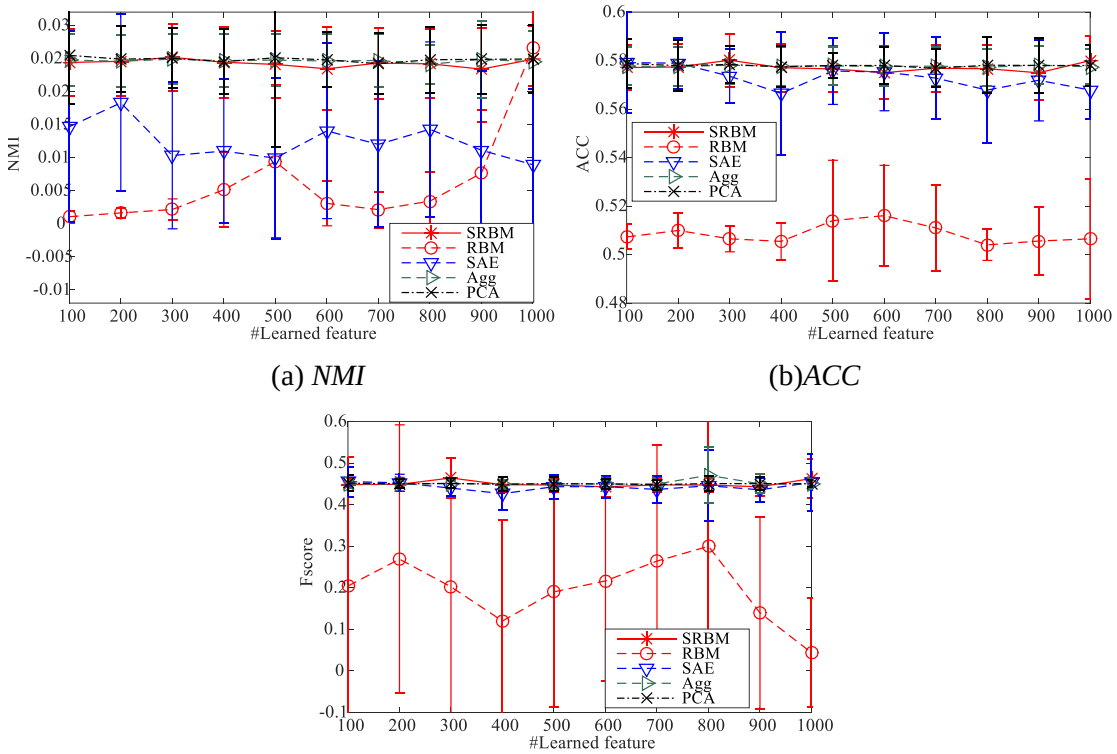


(d) CIC2019

Fig.7 *Fscore* results of different feature learning/extracting algorithms

(3) Performance evaluated on the data with dynamic features

Above experiments are only carried on the dataset with static features. This section evaluates the performance of unsupervised feature learning/extracting methods on the dataset with dynamic features. The *NMI*, *ACC* and *F-score* results of feature learning/extracting methods on the Dynamic dataset are shown in Fig.8. SRBM significantly improves the performance of RBM. And its performance is close to the best one obtained by Agg method.



(a) *NMI* (b)*ACC*

(c)*Fscore*

Fig.8 *NMI*, *ACC* and *Fscore* results on the data with dynamic features

5.2.2 Performance with optimal numbers of features

This section analyzes the performance of unsupervised feature learning/extracting methods with the optimal numbers of features. In the previous section, we have evaluated different unsupervised feature learning/extracting methods while varying the numbers of learned features. In this section, we will evaluate these methods with respect to their optimal numbers of features. Specifically, the number of features that leads to the best performance (with respect to *NMI*) will be adopted for the test method, as the number of features varies from 100 to 1000 with an interval of 100. The *NMI*, *ACC* and *Fscore* results with the optimal number of features are shown in Tables 5 to 7. Origin denotes the results obtained with the full feature set. The optimal numbers of features for different datasets are shown in Table 8.

The results show that the SRBM performs the best on all datasets in terms of *NMI*, *ACC* and *Fscore* except the *NMI* on the Dynamic dataset. In addition, its performance is more stable than that of RBM. For example, RBM obtains much worse performance on the APIPermission than on the API and APIFlowDroid datasets. However, the values of *NMI*, *ACC* and *Fscore* obtained by SRBM on the three OmniDroid datasets with static features are not much different. On CIC2019 dataset, the *Fscore* (0.418) obtained by SRBM is much higher than those obtained by other methods. In addition, using SRBM, the *Fscore* of RBM is improved from 0.085 to 0.418 on the

CIC2019 dataset, and improved from 0.260 to 0.843 on CIC2020 dataset. This further proves that the RBMs trained on the subspaces can benefit by improving its clustering performance.

Table 5 *NMI* results of different methods with the optimal number of learned/extracted features

Datasets	**Origin**	**SRBM**	**RBM**	**SAE**	**PCA**	**Agg**
API	0.175±0.015	**0.193±0.018**	0.192±0.019	0.142±0.073	0.175±0.015	0.176 ±0.015
APIFlowDroid	0.175±0.015	**0.193±0.018**	0.185±0.019	0.106±0.088	0.175±0.015	0.176±0.015
APIPermission	0.176±0.015	**0.198±0.076**	0.083±0.031	0.137±0.070	0.176±0.015	0.177±0.015
CIC2019	0.021±0.021	**0.101±0.069**	0.019±0.025	0.012±0.024	0.053±0.037	0.079±0.061
CIC2020	0.195	**0.265**	0.227	0.112	0.198	0.207
Dynamic	0.025±0.005	0.025±0.005	**0.027±0.006**	0.018±0.013	0.025±0.007	0.025±0.004

Table 6 *ACC* results of different methods with the optimal number of learned/extracted features

Datasets	**Origin**	**SRBM**	**RBM**	**SAE**	**PCA**	**Agg**
API	0.696±0.012	**0.721±0.014**	0.717±0.015	0.694±0.015	0.696±0.013	0.697±0.013
APIFlowDroid	0.696±0.013	**0.720±0.014**	0.709±0.016	0.695±0.015	0.696±0.013	0.697±0.013
APIPermission	0.697±0.013	**0.735±0.054**	0.662±0.039	0.694±0.014	0.697±0.013	0.697±0.014
CIC2019	0.745±0.033	**0.775±0.029**	0.735±0.032	0.748±0.035	0.726±0.076	0.675±0.078
CIC2020	0.623	**0.804**	0.629	0.516	0.629	0.629
Dynamic	0.578±0.010	**0.580±0.011**	0.507±0.025	0.579±0.010	0.579±0.010	0.578±0.008

Table 7 *Fscore* results of different methods with the optimal number of learned/extracted features

Datasets	**Origin**	**SRBM**	**RBM**	**SAE**	**PCA**	**Agg**
API	0.599±0.014	**0.653±0.019**	0.643±0.017	0.595±0.021	0.599±0.014	0.600±0.014
APIFlowDroid	0.599±0.014	**0.651±0.016**	0.628±0.020	0.596±0.020	0.599±0.014	0.600±0.015
APIPermission	0.600±0.015	**0.685±0.034**	0.650±0.038	0.593±0.016	0.600±0.015	0.601±0.016
CIC2019	0.030±0.042	**0.418±0.123**	0.085±0.112	0.067±0.069	0.082±0.057	0.279±0.202
CIC2020	0.579	**0.843**	0.260	0.680	0.773	0.773
Dynamic	0.450±0.013	**0.464±0.049**	0.044±0.131	0.453±0.020	0.452±0.018	0.450±0.010

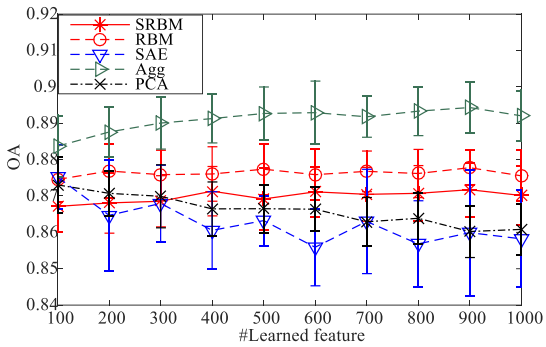
The optimal numbers of learned/extracted features with the best *NMI* obtained by different methods are shown in Table 8. For example, the SRBM with 400, 1000, 100, 400, 1000 and 300 learned features on API, APIFlowDroid, APIPermission, CIC2019, CIC2020 and Dynamic datasets respectively obtain the best *NMI* when compared with other numbers of learned features. And the optimal numbers of features using SRBM are less than those using RBM on most of datasets. This suggests that SRBM requires to learn less features than RBM. This is able to further decrease the time consumption for training malware detection models.

Table 8 The optimal numbers of learned/extracted features with the best *NMI*

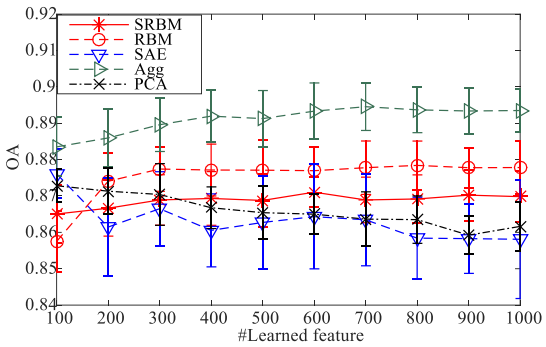
Datasets	**SRBM**	**RBM**	**SAE**	**PCA**	**Agg**
API	400	1000	800	900	700
APIFlowDroid	1000	1000	900	400	600
APIPermission	100	200	500	100	800
CIC2019	400	700	400	900	500
CIC2020	1000	500	400	100	100
Dynamic	300	1000	200	100	300

5.3 Performance evaluated by classification evaluation metrics

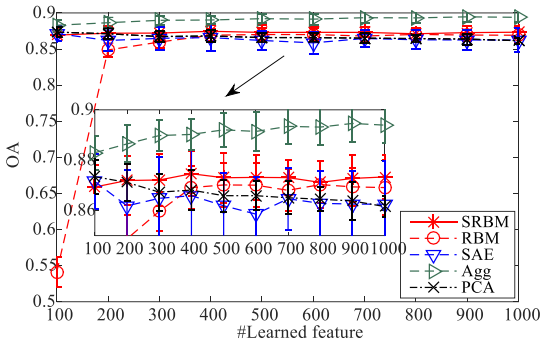
This section evaluates the performance of learned/extracted features in terms of classification evaluation metrics, i.e., *OA* and *F-measure*. In[57], the unsupervised feature learning algorithm is also evaluated using classification algorithms. In this paper, Random Forest is applied as the basic classification algorithm. The *OA* and *F-measure* obtained by these feature learning/extraction algorithms are shown in Figs. 9 and 10 respectively. The results on the CIC2020 datasets are not shown, because the experiments on this dataset aims at illustrating the performance of unsupervised feature learning method used in the case of unsupervised zeroday malware detection. So, it is not evaluated in terms of classification performance.



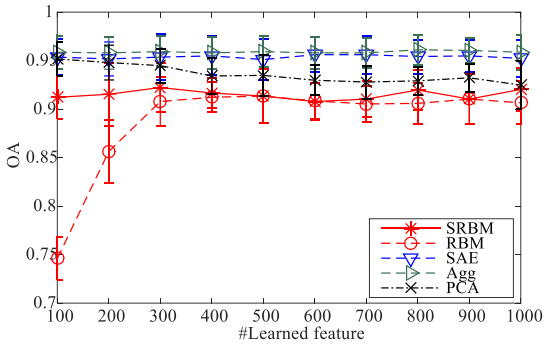
(a) API



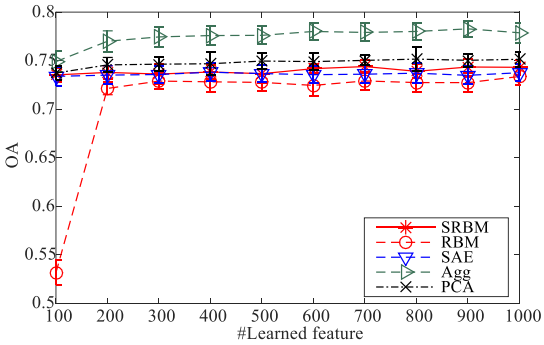
(b) APIFlowdroid



(c) APIPermission



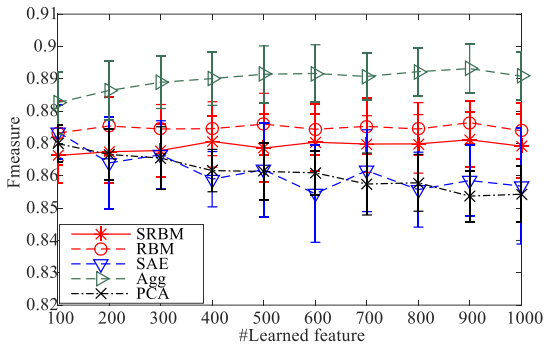
(d)CIC2019



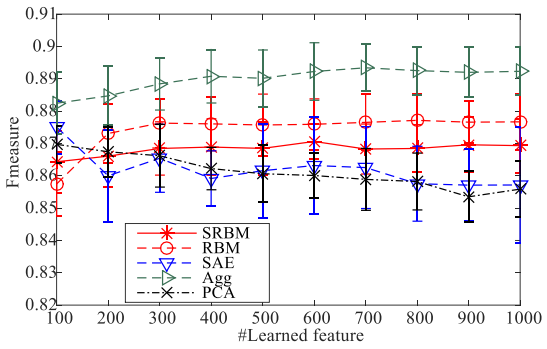
(e) Dynamic

Fig.9 *OA* results of different feature learning/extracting algorithms

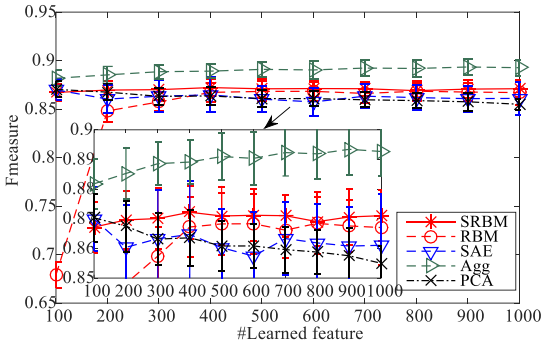
In terms of *OA* and *F-measure*, the SRBM outperforms RBM on APIPermission and CIC2019 datasets. Even though RBM performs better than SRBM on API and APIFlowdroid, the *OAs* and F-*measures* of SRBM are close to those of RBM. In detail, the best *OA* of SRBM is 0.877, and the one of RBM is 0.873; the best *F-measure* of SRBM is 0.876 and that of RBM is 0.872 when evaluated on APIPermission dataset; the best *OA* of SRBM is 0.871, and the one of RBM is 0.878; the best *F-measure* of SRBM is 0.877 and that of RBM is 0.871 when evaluated on API dataset.



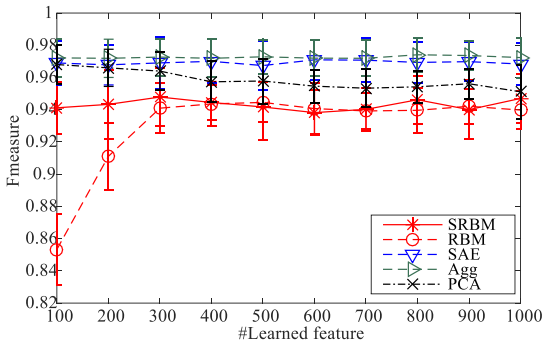
(a) API



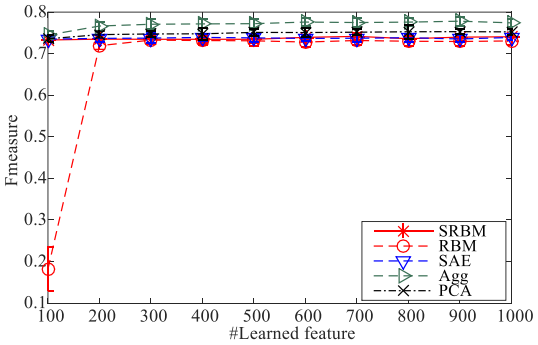
(b) APIFlowdroid



(c) APIPermission



(d)CIC2019



(e) Dynamic

Fig.10 *F-measure* results of different feature learning/extracting algorithms

The *OA* and *F-measure* with the optimal numbers of learned/extracted features are shown in Tables 9 and 10 respectively. The Origin denotes the results obtained by Random Forest with origin full feature set. The results show that the *OA* and *F-measure* are decreased by feature reduction methods when compared with Origin on API and CIC2019 datasets. This is because that the unsupervised feature learning/extracting methods did not use the label information and may lose some information that benefit for classification. The *OA* and *F-measure* are decreased a little bit on API and APIFlowDroid; but these are increased on APIPermission and CIC2019 datasets when SRBM is compared with RBM. In addition, the results in Section 5.4.2 show that the *OA* and *F-measure* of SRBM could be further improved when the number of subspaces is increased. Results show that the *OA* and *F-measure* could be close to that obtained by Agg when the number of subspaces is 1000.

Table 9 OA results of different methods with the optimal number of learned/extracted features

Datasets	Origin	SRBM	RBM	SAE	PCA	Agg
API	**0.893±0.007**	0.871±0.007	0.876±0.007	0.858±0.013	0.867±0.007	0.890±0.007
APIFlowDroid	0.893±0.009	0.870±0.007	0.878±0.007	0.860±0.010	0.865±0.007	**0.893±0.006**
APIPermission	0.894±0.007	0.869±0.007	0.849±0.010	0.893±0.007	0.871±0.008	**0.895±0.007**
CIC2019	**0.960±0.013**	0.916±0.015	0.905±0.017	0.954±0.018	0.952±0.017	0.959±0.017
Dynamic	**0.786±0.008**	0.744±0.007	0.734±0.009	0.738±0.010	0.752±0.012	0.783±0.008

Table 10 F-measure results of different methods with the optimal number of learned/extracted features

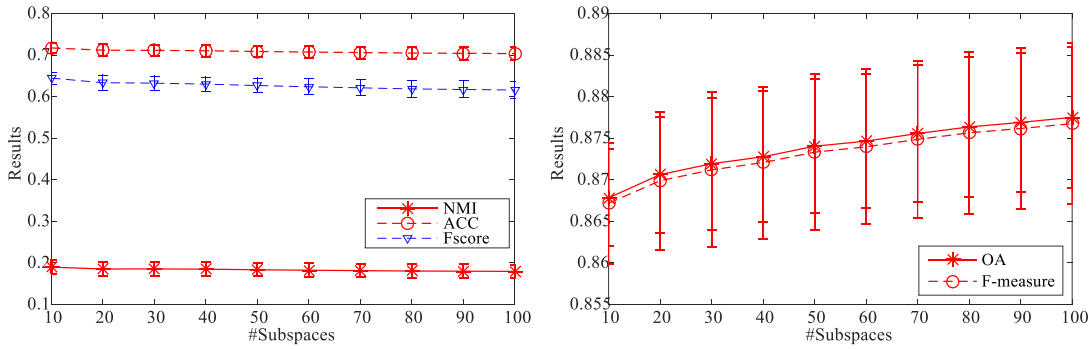
Datasets	Origin	SRBM	RBM	SAE	PCA	Agg
API	**0.892±0.009**	0.870±0.008	0.874±0.009	0.857±0.014	0.861±0.008	0.889±0.008
APIFlowDroid	0.892±0.010	0.869±0.008	0.877±0.009	0.859±0.009	0.861±0.009	**0.893±0.007**
APIPermission	**0.893±0.008**	0.867±0.009	0.848±0.011	0.892±0.008	0.868±0.009	0.893±0.009
CIC2019	**0.973±0.009**	0.945±0.111	0.939±0.111	0.970±0.013	0.968±0.012	0.972±0.012
Dynamic	**0.781±0.006**	0.742±0.009	0.731±0.010	0.739±0.010	0.753±0.017	0.778±0.011

5.4 Discussion

This section mainly discusses the number of subspaces and the clustering method used in our method.

5.4.1 Discussion regarding the number of subspaces

The number of subspaces in SRBM may influence the performance. This section carries out experiments on evaluating the performance of SRBM with the number of subspaces in the range [10,20,30,40,50,60,70,80, 90,100]. The results on the API dataset with different numbers of subspaces are shown in Fig. 11. The results on the other datasets have the same trend. The *NMI*, *ACC* and *Fscore* are decreased a little bit by increasing the number of subspaces. For example, the *NMI* is decreased from 0.192 to 0.179, the *ACC* decreased from 0.720 to 0.703, and *Fscore* decreased from 0.652 to 0.616. The *OA* and *F-measure* are increased by increasing the number of subspaces. The *OA* is increased from 0.869 to 0.878 and *F-measure* increased from 0.867 to 0.876.

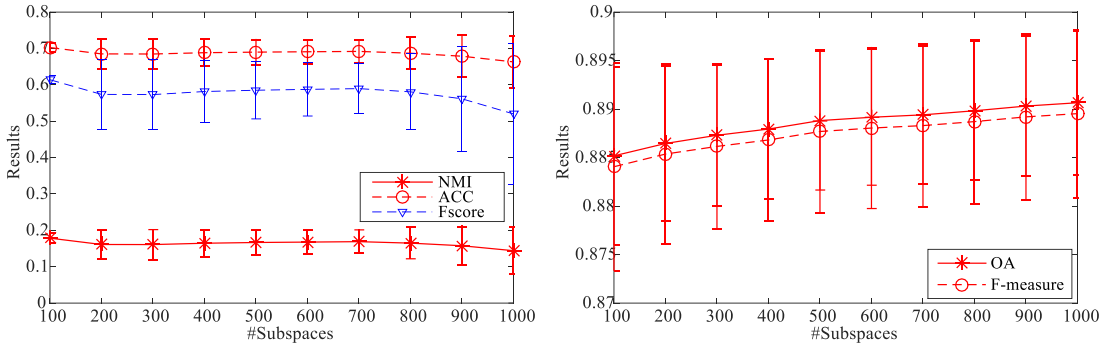


(a) *NMI*, *ACC* and *Fscore* metrics (b) *OA* and *F-measure* metrics

Fig. 11 Discussion on the number of subspaces used in SRBM with the number of 100 learned features

When the number of learned features is 1000, the number of subspaces is set in the range [100,200, 300,400,500,600,700,800,900,1000]. The results are shown in Fig.12. The trend of these performance evaluation metrics with increasing the number of subspaces is the same as those shown in Fig.11. In addition, the SRBM can obtain about 0.89 *OA* and 0.89 *F-measure* when the number of subspaces is 1000 with 1000 learned features. That is close to the Origin

(0.893) shown in Table 9. This suggests that the classification performance could be further improved by increasing the number of subspaces.

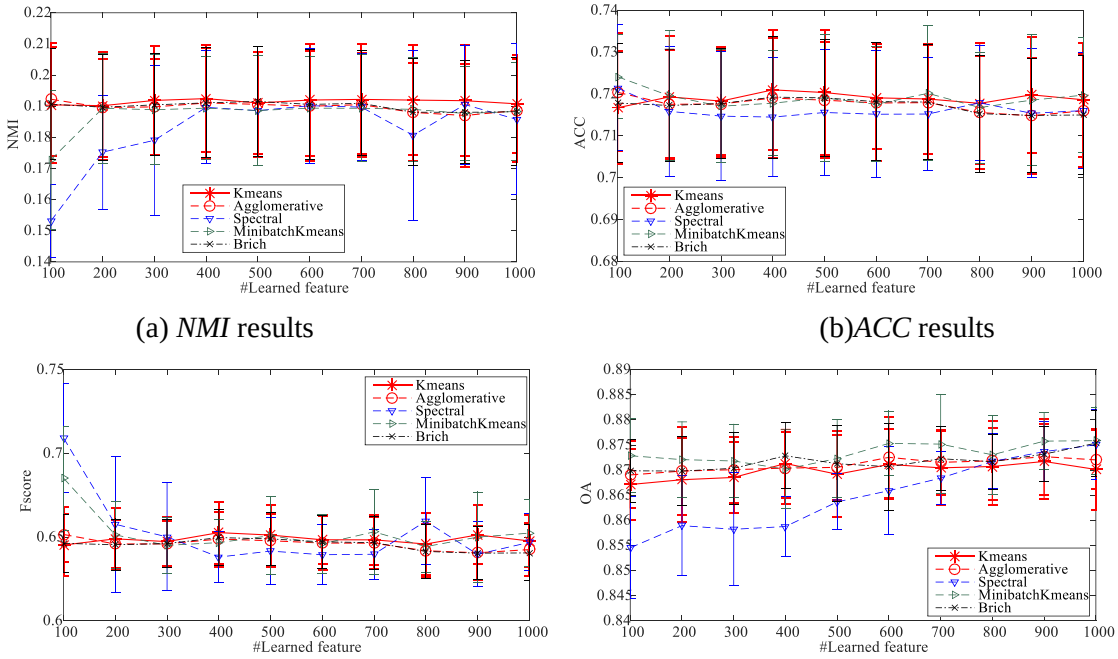


(a) *NMI*, *ACC* and *Fscore* metrics (b) *OA* and *F-measure* metrics

Fig. 12 Discussion on the number of subspaces used in SRBM with the number of 1000 learned features

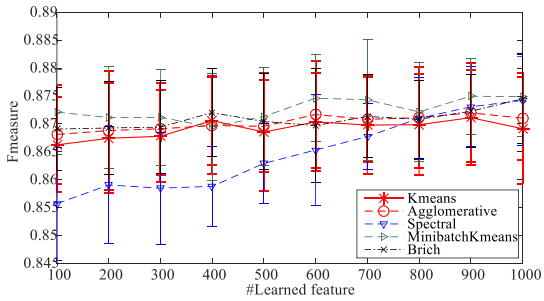
5.4.2 Discussion on the clustering methods

To find the subspaces in SRBM, the K-means algorithm, which is popular for clustering, is applied in this paper. This section will carry out experiments to evaluate the other clustering methods for finding the subspaces. AgglomerativeClustering, SpectralClustering, MinibatchKmeans and Birch algorithms in the Sklearn module are evaluated in this section. The results obtained by SRBM with different clustering algorithms on the API dataset are shown in Fig.13.



(a) *NMI* results (b)*ACC* results

(c) *Fscore* results (d) *OA* results



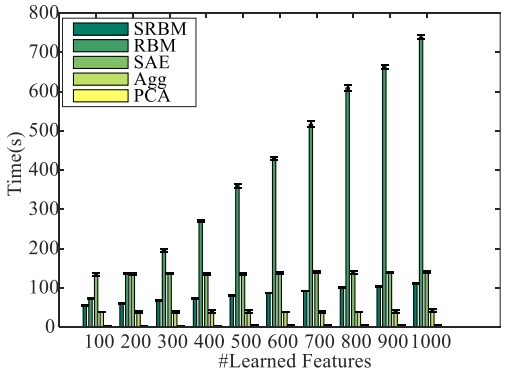
(e) *F-measure* results

Fig.13 Discussion on the clustering methods used in SRBM

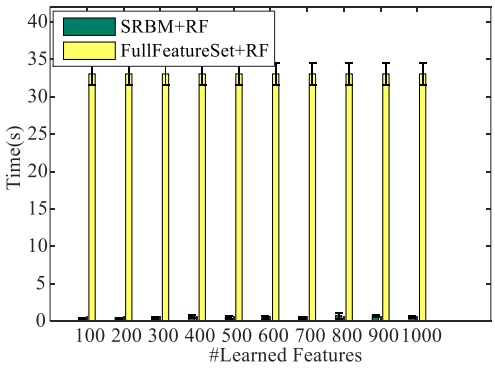
The K-means algorithm is that used in the earlier experiments. The results show that the SRBM with K-means obtains the best results in most cases where the performance is evaluated by the clustering metrics, i.e., *NMI*, *ACC* and *Fscore*. Its performance is more stable than that of other methods as the number of learned features changes. For example, the Spectral and MiniBatchKmeans obtain about 0.71 *Fscore* with 100 learned features, but *Fscore* drops to 0.65 with 200 learned features. The *NMI, ACC, Fscore, OA and F-measure* obtained by the SRBM with K-means does not fluctuate significantly.

5.4.3 Discussion on the time consumption

This section analyzes the time consumption of feature learning and classification model training. In order to evaluate the overhead of different feature learning/extraction methods, taking API data as an example, the time consumed for learning or extracting features by different methods is shown in Fig.14(a). The experiments are carried out on a server with Intel(R) Xeon(R) Gold 6142 CPU and 192GB memory. Each experiment is repeated 10 times. The time consumed by RBM increases with the increase of learned features. This is because the number of parameters required to be solved increases with the increase of learned features. In addition, the results show that SRBM consumes much less time than RBM. This demonstrates that SRBM can decrease the resource consumption of RBM.



(a) Time consumption of feature learning/extracting methods



(b) Time consumption of training Random Forest model

Fig.14 Time consumption results

Tables 9 and 10 show that the Random Forest with the full feature set obtains the best *OA* and *F-measure* on 3 over 5 datasets. This suggests that the feature reduction may degrade classification accuracy. Here is a question: if it is worth to perform feature reduction in such case? One benefit of feature reduction is that it could reduce the time consumption of training

classification models. So, we further looking at the training time consumption of using Random Forest to illustrate if it is worth to perform feature reduction. Taking API dataset as an example, the time consumption results are shown in Fig.14(b). SRBM+RF denotes the time consumption of performing SRBM on the training data and that of training Random Forest model on the training data with reduced feature set. We did not add the training time of SRBM model. This because that we only need to perform the trained SRBM model and train the classification model on the future data to obtain the classification model once the SRBM model is already trained. FullFeatuareSet+RF denotes the time consumption of training Random Forest model on the data with full feature set. The results show that the time consumption of SRBM+RF is much less than that of FullFeatureSet+RF. This further proves that the trained SRBM model is able to reduce the time consumption on training Random Forest model.

6. Conclusions and future work

This paper proposes an unsupervised feature learning method for mobile malware detection. It is based on RBM and it could be used for reducing the data dimensionality in the case of unsupervised mobile malware detection. In order to decrease resource consumption and to improve the performance of RBM, it optimizes RBM by introducing the subspaces concept. Before feature learning, a search for appropriate subspaces is conducted using a clustering method on the full feature set, and an RBM is used for learning the features under each feature subspace. Finally, all learned features are combined to represent origin data in lower dimension. The SRBM is compared with RBM, SAE, Agg and PCA in our experiments. The results show that SRBM performs better than other methods on most of datasets in terms of clustering evaluation metrics especially in the case of zeroday malware detection. And, SRBM improves the *NMI*, *ACC* and *Fscore* of RBM. On average, the *NMI*, *ACC* and *Fscore* are respectively improved about 6.2%, 6.9% and 15.4% over all datasets. In addition, its performance is more stable than that of other methods.

SRBM is able to improve the performance of RBM, but the performance is still not good on some datasets. In the future, we plan to study the method for improving the performance of unsupervised feature learning method in the field of mobile malware detection. And we will also study the multimodal feature learning algorithms on hybrid features, aiming at decreasing the number of hybrid features in an unsupervised way for malware detection. And the subspaces division will be further studied in other methods such as the SAE etc. in future.

Acknowledgements

We thank the anonymous reviewers for their constructive comments. This work is supported by the National Natural Science Foundation of China [Grant No. 61501128, 61976239], financial support from China Scholarship Council, Natural Science Foundation of Guangdong Province [Grant Nos. 2017A030313345, 2020A1515010783].

References

[1] Hou, S. F., Ye, Y. F., Song, Y. Q., et al. HinDroid: An intelligent android malware detection system based on structured heterogeneous information network. the 23rd ACM International Conference on SIGKDD, 2017: 1-9.

[2] 10 mobile usage statistics every marketer should know in 2021. https://www.oberlo.com/blog/mobile-usage-statistics, Last accessed: 2021-01-05.

[3] What is mobile malware? https://www.crowdstrike.com/epp-101/mobile-malware/, Last accessed: 2021-01-05.

[4] AV-Test malware report, https://www.av-test.org/en/statistics/malware/, Last accessed: 2021-01-05.

[5] 2019 Kaspersky Cybersecurity Report, https://www.techarp.com/cybersecurity/2019-kaspersky-cybersecurity-report/, Last accessed: 2021-01-05.
[6] G DATA Mobile Malware Report 2019: New high for malicious Android apps, https://www.gdatasoftware.com/news/g-data-mobile-malware-report-2019-new-high-for-malicious-android-apps, Last accessed: 2021-01-05.
[7] Chen, Z., Yan, Q., Han, H., et al. Machine learning based mobile malware detection using highly imbalanced network traffic. Information Sciences, 2018, 433-434: 346-364.
[8] Jain, A. K., Gupta B. B. A machine learning based approach for phishing detection using hyperlinks information. Journal of Ambient Intelligence & Humanized Computing, 2019, 10(5): 2015-2028.
[9] Al-Sharif, Z. A., Al-Saleh, M. I., Alawneh L., et al. Live forensics of software attacks on cyber-physical systems. Future Generation Computer Systems. 2020, 108: 1217-1229.
[10]Perera, C., Barhamgi, M., Bandara, A. K., et al., Designing privacy-aware internet of things applications. Information Sciences, 2020, 512: 238-257.
[11]Ouaguid, A., Abghour, N., Ouzzif, M. A novel security framework for managing android permissions using blockchain technology. International Journal of Cloud Applications and Computing,2018, 8(1): 55-79.
[12]Azad, M. A., Arshad, J., Kamal, S. M. A., et al. A first look at privacy analysis of COVID-19 contact tracing mobile applications. IEEE Internet of Things Journal, 2020: 1-11.
[13]Tam, K., Feizollah, A., Anuar, N. B., et al, The evolution of android malware and android analysis techniques. ACM Computing Surveys, 2017, 49(4): 76:1-76:41.
[14]Zheng, M., Sun, M., Lui, J. C. S.. DroidAnalytics: A signature based analytic system to collect, extract, analyze and associate android malware. Proceedings of 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, 2013: 163-171.
[15]Sharma, K., Gupta, B. B. Mitigation and risk factor analysis of android applications. Computers & Electrical Engineering, 2018, 71: 416-430.
[16]Seo, S.H., Gupta, A., Sallam, A.M., et al. Detecting mobile malware threats to homeland security through static analysis. Journal of Network and Computer Applications, 2014, 38: 43-53.
[17]Yuan, Z., Lu, Y., Wang, Z., et al. Droid-sec: deep learning in Android malware detection. In Proceedings of the 2014 ACM Conference on SIGCOMM, 2014: 371-372.
[18]Potharaju, R., Newell, A., Nita-Rotaru, C., et al. Plagiarizing smartphone applications: attack strategies and defense techniques. ACM International Symposium on Engineering Secure Software and Systems, 2012: 106-120.
[19]Xiao, X., Xiao, X., Jiang, Y. et al. Identifying Android malware with system call co-occurrence matrices. Transactions on Emerging Telecommunications Technologies, 2016, 27: 675-684.
[20]Thiyagarajan, J., Akash, A., Murugan, B., Improved real-time permission based malware detection and clustering approach using model independent pruning, IET Information Security, 2020, 14(5): 531-541.
[21]Pai, S., Troia, F. D., Visaggio, C. A., et al. Clustering for malware classification. Journal of Computer Virology and Hacking Techniques, 2017, 13(2): 95-107.
[22]Yerima, S. Y., Sezer, S. DroidFusion: A novel multilevel classifier fusion approach for Android malware detection, IEEE Transactions on Cybernetics, 2018, 49(2): 453-466.
[23]Taheri, R., Ghahramani, M., Javidan, R., et al. Similarity-based Android malware detection using Hamming distance of static binary features. Future Generation Computer Systems, 2020, 105(2020): 230-247.
[24]Badhani, S., Muttoo, S. K., CENDroid-A cluster-ensemble classifier for detecting malicious Android applications. Computers&Security, 2019, 85(2019): 25-40.
[25]Duc, N. V., Giang, P. T., NADM: Neural network for android detection malware. The 9th International Symposium on Information and Communication Technology, 2018: 449-455.
[26]Karbab, E. B., Debbabi, M., Derhab, A. et al. MalDozer: Automatic framework for android malware detection using deep learning. Digital Investigation, 2018, 24 (2018): 48-59.
[27]Mercaldo, F., Santone, A., Deep learning for image-based mobile malware detection, Journal of Computer Virology and Hacking Techniques, 2020, 16(6): 1-15.

[28]Alam, S., Alharbi, S. A., Yildirim, S. Mining nested flow of dominant APIs for detecting android malware. Computer Networks, 2020, 167 (2020): 1-10.
[29]Mariconti, E., Onwuzurike, L., Andriotis, P., et al. MaMaDroid: Detecting android malware by building markov chains of behavioral models. Network and Distributed System Security Symposium, 2017:1-16.
[30]Yen, Y. S., Sun, H. M. An Android mutation malware detection based on deep learning using visualization of importance from codes. Microelectronics Reliability, 2019, 93(2019): 109-114.
[31]Ananya, A., Aswathy, A., Amal, T. R., et al. SysDroid: a dynamic ML-based android malware analyzer using system call traces. Cluster Computing, 2020:1-20.
[32]Duarte-Garcia, H. L., Cortez-Marquez, A., Sanchez-Perez, G., et al. Automatic malware clustering using word embeddings and unsupervised learning, IAPR/IEEE International Workshop on Biometrics and Forensics, 2019: 1-6.
[33]Shamsi, F. A., Woon W. L., Aung, Z., Discovering similarities in malware behaviors by clustering of API call sequences. International Conference on Neural Information Processing, 2018: 122-133
[34]Pang, Y., Peng, L. Z., Chen, Z. X. et al. Imbalanced learning based on adaptive weighting and Gaussian function synthesizing with an application on Android malware detection. Information Sciences, 2019, 484(2019): 95-112.
[35]Angelo, G., Ficco, M., Palmieri, F. Malware detection in mobile environments based on Autoencoders and API-images. Journal of Parallel and Distributed Computing, 2020, 137(2020): 26-33.
[36]Xiao, X., Zhang, S. F., Mercaldo, F., et al. Android malware detection based on system call sequences and LSTM. Multimedia Tools and Applications, 2019, 78(4): 3979-3999.
[37]Martin, A., Lara-Cabrera, R., Camacho, D., Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the OmniDroid dataset. Information Fusion, 2019, 52 (2019): 128-142.
[38]Scalas, M., Maiorca, D., Mercaldo, F., On the effectiveness of system API-related information for Android ransomware detection. Computers&Security, 2019, 86(2019): 168-182.
[39]Taheri, L., Kadir, A. F. A., Lashkari, A. H. Extensible android malware detection and family classification using network-flows and API-Calls. 2019 International Carnahan Conference on Security Technology, 2019: 1-9.
[40]Alzaylaee, M. K., Yerima, S. Y., Sezer, S. DL-Droid: Deep learning based android malware detection using real devices, Computers & Security, 89(2020)101663.
[41]Saif, D., EI-Gokhy, S. M., Sallam, E., Deep belief networks-based framework for malware detection in Android systems. Alexandria Engineering Journal, 2018, 57(2018): 4049-4057.
[42]Feizollah, A., Anuar, N. B., Salleh, R., et al. A review on feature selection in mobile malware detection. Digital Investigation, 2015,13: 22-37.
[43]Suresh, S., Di, Troia F., Potika, K., et al. An analysis of android adware. Journal of Computer Virology and Hacking Techniques, 2019, 15:147-160.
[44]Lashkari, A. H., Kadir, A. F. A., Taheri, L. G., Toward developing a systematic approach to generate benchmark android malware datasets and classification, Proc. of IEEE International Carnahan Conference on Security Technology (ICCST), 2018: 1-7.
[45]Sheen, S., Anitha, R., Natarajan, V. Android based malware detection using a multifeature collaborative decision fusion approach. Neurocomputing, 2015, 151: 905-912.
[46]Srivastava, N., Salakhutdinov, R. Multimodal learning with deep boltzmann machines. Journal of Machine Learning Research, 2012, 15:1-32.
[47]Smolensky, P. Information processing in dynamical systems: Foundations of harmony theory. In Parallel Distributed Processing: Explorations in the Microstructure of Cognition, 1986, 1(1986): 194-281.
[48]Fischer, A., Igel, C. An introduction to restricted boltzmann machines. Iberoamerican Congress on Pattern Recognition. 2012:14-36.
[49]Carreira-Perpinan, M.A., Hinton, G.E. On contrastive divergence learning. Aistats 2005, 10: 33-40.
[50]Gu, Q., Li, Z., Han, J., Joint feature selection and subspace learning, Proc. of International Joint

Conference on Artificial Intelligence, IJCAI, 2011: 1294-1299.
[51]Du, L., Shen Y. D., Unsupervised feature selection with adaptive structure learning, International Conference on Knowledge Discovery and Data Mining, 2015: 209-218.
[52]Huang, D., Cai, X., Wang, C. D. Unsupervised feature selection with multi-subspace randomization and collaboration, Knowledge-Based Systems, 182 (2019): 104856.
[53]Arzt, S., Rasthofer, S., Fritz, C., et al. Flowdroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps, in: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, 2014: 259–269.
[54]Rahali, A., Lashkari, A. H., Kaur G., et al., DIDroid: Android Malware Classification and Characterization Using Deep Image Learning, 10th International Conference on Communication and Network Security, 2020.
[55]CCCS-CIC-AndMal-2020, https://www.unb.ca/cic/datasets/andmal2020.html, Last accessed: 2021-01-05.
[56]Zhou, P., Chen, J., Fan, M., et al. Unsupervised feature selection for balanced clustering. Knowledge Based Systems. 2020, 193: 105417.
[57]Yan, X., Nazmi, S., Erol, B. A., et al. An Efficient Unsupervised Feature Selection Procedure Through Feature Clustering. Pattern Recognition Letters, 2020, 131: 277-284.
[58]Telikani, A., Gandomi, A. H. Cost-sensitive stacked auto-encoders for intrusion detection in the Internet of Things, 2019: 1-19 (published online).
[59]Ng., W.W., Zeng, G., Zhang, J., et al. Dual autoencoders features for imbalance classification problem, Pattern Recognition, 60 (2016): 875-889.
[60]Gormley, M., Principal Component Analysis and Dimensionality Reduction, https://www.cs.cmu.edu/~mgormley/courses/10701-f16/slides/lecture14-pca.pdf, Last accessed: 2021-01-05.
[61]Agglomeration Feature Selection, https://scikit-learn.org/stable/modules/generated/sklearn.cluster.FeatureAgglomeration.html, Last accessed: 2021-01-05.
[62]Scikit-learn, https://scikit-learn.org, Last accessed: 2021-01-05.
[63]Wenbin Zhang and Eirini Ntoutsi. Faht: an adaptive fairness-aware decision tree classifier. In International Joint Conference on Artificial Intelligence (IJCAI), pages 1480–1486, 2019.
[64]Wenbin Zhang, Xuejiao Tang, and Jianwu Wang. On fairness-aware learning for non-discriminative decision-making. In International Conference on Data Mining Workshops (ICDMW), pages 1072–1079, 2019.
[65]Wenbin Zhang and Albert Bifet. Feat: A fairness-enhancing and concept-adapting decision tree classifier. In International Conference on Discovery Science, pages 175–189. Springer, 2020.
[66]Wenbin Zhang et al. Flexible and adaptive fairness-aware learning in non-stationary data streams. In IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI), pages 399–406, 2020.
[67]Wenbin Zhang and Liang Zhao. Online decision trees with fairness. arXiv preprint arXiv:2010.08146, 2020.
[68]Wenbin Zhang. Learning fairness and graph deep generation in dynamic environments. 2020.
[69]Wenbin Zhang, Albert Bifet, Xiangliang Zhang, Jeremy C Weiss, and
[70]Wolfgang Nejdl. Farf: A fair and adaptive random forests classifier. In Pacific-Asia Conference on Knowledge Discovery and Data Mining, pages 245–256. Springer, 2021.
[71]Wenbin Zhang and Jeremy Weiss. Fair decision-making under uncertainty. In 2021 IEEE International Conference on Data Mining (ICDM). IEEE, 2021.
[72]Wenbin Zhang and Jeremy C Weiss. Longitudinal fairness with censorship. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 36, pages 12235–12243, 2022.
[73]Wenbin Zhang, Shimei Pan, Shuigeng Zhou, Toby Walsh, and Jeremy C Weiss. Fairness amidst non-iid graph data: Current achievements and future directions. arXiv preprint arXiv:2202.07170, 2022.
[74]Wenbin Zhang, Tina Hernandez-Boussard, and Jeremy C Weiss. Censored fairness through awareness. In Proceedings of the AAAI Conference on Artificial Intelligence, 2023.
[75]Wenbin Zhang and Jeremy Weiss. Fairness with censorship and group constraints. Knowledge and

Information Systems, 2022.
[76]Wenbin Zhang and Jianwu Wang. A hybrid learning framework for imbalanced stream classification. In IEEE International Congress on Big Data (BigData Congress), pages 480–487, 2017.
[77]Wenbin Zhang, Jian Tang, and Nuo Wang. Using the machine learning approach to predict patient survival from high-dimensional survival data. In IEEE International Conference on Bioinformatics and Biomedicine (BIBM), 2016.
[78]Wenbin Zhang and Jianwu Wang. Content-bootstrapped collaborative filtering for medical article recommendations. In IEEE International Conference on Bioinformatics and Biomedicine (BIBM), 2018.
[79]Xuejiao Tang, Liuhua Zhang, et al. Using machine learning to automate mammogram images analysis. In IEEE International Conference on Bioinformatics and Biomedicine (BIBM), pages 757–764, 2020.
[80]Mingli Zhang, Xin Zhao, et al. Deep discriminative learning for autism spectrum disorder classification. In International Conference on Database and Expert Systems Applications, pages 435–443. Springer,2020.
[81]Wenbin Zhang, Jianwu Wang, Daeho Jin, Lazaros Oreopoulos, and Zhibo Zhang. A deterministic self-organizing map approach and its application on satellite data based cloud type classification. In IEEE International Conference on Big Data (Big Data), 2018.
[82]Xuejiao Tang, Xin Huang, et al. Cognitive visual commonsense reasoning using dynamic working memory. In International Conference on Big Data Analytics and Knowledge Discovery. Springer, 2021.
[83]Wenbin Zhang, Liming Zhang, Dieter Pfoser, and Liang Zhao. Disentangled dynamic graph deep generation. In Proceedings of the SIAM International Conference on Data Mining (SDM), pages 738–746, 2021.