國立政治大學資訊管理學系研究所

碩士學位論文

二元主體學習技術研究與張量流實作

Bipartite Majority Learning with Tensors

指導教授：　郁方　博士

研究生　：　李佳倫　撰

中華民國　一〇八年　一月

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

Abstract

A great deal of attention has been given to machine learning owing to the remarkable achievement in Go game and AI robot. Since then, machine learning techniques have been widely used in computer vision, information retrieval, and speech recognition. However, data are inevitably containing statistically outliers or mislabeled. These anomalies could interfere with the effectiveness of learning. In a dynamic environment where the majority pattern changes, it is even harder to distinguish anomalies from majorities. This work addresses the research issue on resistant learning on categorical data. Specifically, we propose an efficient bipartite majority learning algorithm for data classification with tensors. We adopt the resistant learning approach to avoid significant impact from anomalies and iteratively conduct bipartite classification for majorities afterward. The learning system is implemented with TensorFlow API and uses GPU to speed up the training process. Our experimental results on malware classification show that our bipartite majority learning algorithm can reduce training time significantly while keeping competitive accuracy compared to previous resistant learning algorithms.

Keywords: Bipartite majority learning, Resistant learning, Malware classification.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

Contents

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

List of Figures

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

List of Tables

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

1 Introduction

Artificial Neural Network (ANN) is regularly adopted in data classification researches [1, 2, 3, 4]. These studies combine the softmax transformation in the output layer of ANNs so that the output values of each output nodes can represent the probability of a specific input belongs to that class. With the characteristics that ANN is suitable for classification, ANN can also assist humans in solving yes-no questions. For example, determining if the image contains a specific object, forecasting whether future stock prices will rise, etc. A complex ANN has been proved that it could represent any input-output mappings [5]. If sufficient information was supplied, ANN could help people making decisions more effectively. However, the parameters of ANN need many tests to find proper settings. How many hidden layers should be adopted? How many hidden nodes in each layer? Long trial and error processes might consume a lot of time and computing resources. Even if a proper model complexity is found, the loss of the model is often stuck in local optimal and might not fulfill the requirement of the classification accuracy on training data. Still, in real-world data sets, data is inevitably containing outliers or wrong information. Although ANN is more flexible to handle mislabeled or outliers rather than linear models [6], a small portion of outliers will still affect the learning effect of the majority. Tsaih and Chang [7] proposed a resistant learning procedure on ANN. The learning model of resistant learning is single-hidden layer feed-forward neural networks (SLFN). This resistant learning method allows multi-dimension inputs and only one dimension output. In other words, the SLFN can have many input nodes but can have only one output node. The resistant learning method dynamically changes the hidden node amount to find a proper model complexity. When ANN's loss stuck in local optimal, resistant learning approach will add new hidden nodes and calculate the appropriate weights for the new hidden nodes. With this mathematical approach, the data which is difficult to learn can be learned by the ANN while the majority's numeric output would not be affected significantly. But, the final goal of resistant learning is to find a near perfect fitting-function, that is to say, the resistant learning method focuses on finding perfect numeric input-output mappings, not

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

to be a classifier for data. Still, if ANN learns the perfect input-output relationship of the training data, it would probably cause the over-fitting problem. Therefore, in this paper we designed a new resistant learning method. We retained the resistant learning advantages and applied it to classify different types of data.

The proposed resistant learning mechanism we called bipartite majority learning (BML). Our method can learn the proper majority features of two categories of data. Bipartite classification problem is relative. Although resistant learning method only has one output value of each input data, we can set different desired learning output for two types of data. One of them should be greater, and the other should be smaller. Then, we can find a boundary for classifying different types of data. BML can also pick out the anomalies while the anomalies' features were unknown previously. In order to achieve this goal, BML starts learning from a small portion of training data, then gradually increase the selected amount of training data until the majority features of data are learned. When selecting the partial training data, the whole training data should be sort by specific criteria. This sort-and-select process imitates the learning process of human beings. Humans use things they already know to describe unknown things, and learn the concept of unknown things in the process of such analogy and thinking. By the sorting process, ANN can learn the data which is familiar to it first. As the knowledge of ANN becomes more and more abundant, ANN can distinguish which data belongs to the majority pattern, and those that are difficult to learn may be outliers. In the situation of classifying two types of data, BML would learn the most different features of the bipartite data first. The most different features mean the data that is far from the classification boundary. Then, BML will tune the weights or change the model structure to learn the difference of the similar data. The stop condition of BML is the majority of data can be correctly classified by the boundary. The majority rate can be determined previously. Although BML can even perform perfect classification for all training data due to the resistant learning process, perfect learning might cause an over-fitting problem. Depending on the application, the majority rate can be set as needed. More precisely, the setting of the majority rate should

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

depend on the knowledge of training data. If we know how much outliers may be in the training data, we can set the majority rate and then BML will automatically pick out the appropriate majorities and learn these majorities perfectly.

In the experiments, we apply BML to do malware dynamic analysis. Dynamic analysis refers to the process of executing a program and recording its behavior. The program execution data sets used in our experiments are real-world data sets. The program execution data is a collection of profiles generated from the dynamic analysis. When a program is executed, the monitor records all the function calls (e.g., system call or Windows API call) invoked by the program and saves the call sequence in the profile [8]. We collect several benign programs and malicious programs, then label their profiles as benign (B) or malicious (M) for bipartite classification.

Our paper has three main contributions. First, we proposed BML to deal with bipartite classification problem in the context of resistant learning. Past studies [7, 9] focus on learning a fitting function of the numerical outputs. BML learns a fitting function to be the classifier of two categories of data. The proposed resistant learning method in this paper can be applied to any bipartite data.

Second, BML can find anomalies in a global view and do not need prior knowledge of the training data set. BML has a sort-and-select mechanism for all of the training data, selecting appropriate majority data for model training, decrease the effect from the outliers. This function can be helpful when doing dynamic malware analysis. As we know, a malicious program may not always do something harmful to the infected system (i.e., in the incubation phase); therefore, some function call sequences invoked by the malware do not belong to the attack behavior (which can be considered as a noise or outlier). In this case, the conventional ANN might be confused with the outlier data while training the model. However, the proposed majority learning mechanism for nominal classification can avoid being affected by such anomalies.

Third, BML has greater time efficiency in training the model and keep high prediction accuracy. Compare to the former resistant learning methods, our BML is outperformed in

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

decreasing training time and remains a competitive classification accuracy. Our method can be faster than former ones due to two reasons. One reason is that, we design a looser learning goal for the SLFN model. Former resistant learning methods need to learn a near-perfect input-output relationship for training data. But, in our approach, we only need to let the model classify the outputs of the two categories of data. For example, the model does not need to train all class 1 data to the value of 1 and train all class 2 data to the value of 0, instead, BML determine a boundary to classify the two categories of data. It means that we only need to promise all class 1 predict outputs are greater than the class 2 predict outputs. Another reason is that, the system and the underlying neural network are implemented in Python and TensorFlow API [10] on GPU, which provides the high performance of parallel computing mechanism on the high dimension tensors. A tensor is a set of primitive values shaped into an array of any number of dimensions. TensorFlow can perform parallel computing to large tensors, and also be able to speed up computing by using GPU. We had tested the execution speed on a portable computer, we tried to do a forward pass through a neural network by a tensor with the shape of (1187842, 52). It took 9.998 seconds for JAVA, and 0.273 seconds for TensorFlow in the same hardware environment. It was about 36 times faster by using TensorFlow with GPU calculating. We used TensorFlow as the main framework in order to reduce the execution time for training the SLFN model.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

2 Related Work

Our main learning architecture of bipartite majority learning is SLFN. The SLFN has been discussed in many researches. Huang et al. [11] have proved the ability of SLFN classifying disjoint decision regions with arbitrary shapes in multidimensional cases. They also mention that failures in SLFN classification can be attributed to inadequate learning and inadequate amount of hidden nodes. Tsaih [12] uses SLFN to conduct two class data classification but having a limitation that input data must be binary inputs. Huang et al. [13] develop an efficient learning algorithm, extreme learning machine (ELM), for the SLFN. Feng et al. [14] apply ELM and have a growing hidden node approach on the SLFN. Our approach also has a growing hidden node method for SLFN and focus on two class data classification, but, we do not have the limitation on the input data, the inputs in our study can be real numbers.

The proposed BML can deal with the anomalies in learning process. In the context of linear regression analysis, there are two ways to dealing with outlier problems: deletion diagnostics and robust estimators [15]. One way of deletion diagnostics is to determine the observations that cause the largest change in some regression quantity [16, 17] when they are excluded from the fitting procedure. As for the robust method, one robustness analysis is to focus on trimmed sum of squared residuals instead of including all the squared residuals as in the least squares estimator [18]. BML inspired by this robust approach, limit the attention to a trimmed set of data and gradually increase the subset size. In this way, BML can pick out the appropriate majority and fight against the outliers.

There are many studies focusing on robust learning methods and dealing with anomaly pattern. Ren et al. [19] proposed a robust softmax regression for multi-class classification to cope with noisy data and statistical outliers. Jiang et al. [20] worked on a single layer robust autoencoder. Zhou and Paffenroth [21] devised a robust mechanism in deep autoencoder to find anomalies. Zhao and Fu [22] proposed a robust graph representation method to clearly split the elements in the segmentation of videos. Wang and Tan [23] studied a robust distance metric learning method which distinguishes labeled image data.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

Jia and Zhao [24] worked on deep neural network to develop a Chinese pinyin typo detection system. Our study was inspired by the resistant learning method on SLFN proposed by Tsaih and Chang [7] and make appropriate modifications to make our method more in line with the bipartite classification situations.

This paper is an extension study in the context of resistant learning. The word resistant has the same meaning to robust. In the statistical literature, the terms robust and resistant are often used interchangeably but sometimes have specific meanings [25]. Robust procedure means that the results are not impacted significantly by violations of the model assumptions (i.e., the errors are normally distributed). As for resistant procedures, those whose numerical results are not impacted significantly by outlying observations. Tsaih and Chang [7] proposed a resistant learning procedure on ANN to learn near-perfect real-number input-output mappings. The error values of all training observations will less than a tiny value, ϵ (say, 10^{-6}). However, in order to train a perfect fitting function, the resistant learning procedure suffers high model complexity and long training time. Srivastava et al. [26] introduced dropout nodes in a neural network to reduce computation requirements and hence speed up ANN training. Huang et al. [9] proposed an envelope module to ease the restriction of ϵ. The envelope method covers the fitting function with an envelope and the learning goal is to make majority data covered by the range of envelope. The envelope method allows observations having larger error values, thus accelerate the resistant learning procedure. Yet, past researchers do not address the learning algorithm with nominal anomalies (i.e., outliers) in the context of resistant learning. Hence, we propose a new resistant majority learning mechanism for nominal classification that can antagonize the anomalies in the training data automatically with less training time and high prediction accuracy.

In the field of malware detection, many research studies adopted machine learning techniques. Hou et al. [27] developed a system to learn the features of malicious Android application API calls. Grosse et al. [28] and Wang et al. [29] proposed adversary approaches in the deep neural network. Grosse et al. [28] used the deep neural network as a

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

two-class classifier for static malware analysis. Wang et al. [29] developed a deep neural network on audit logs to perform dynamic malware analysis. They also demonstrated the validity of the algorithm in other image data sets. Dahl et al. [30] learned the API calls feature and adopt random projections to deal with large, sparse, binary feature sets. Chiu et al. [31] performed clustering for system call sequences and found the features of malware behaviors. Training SLFN to distinguish between malicious behavior and benign behavior is a good application of the bipartite classification. Therefore, we decided to test the detection rate of malicious behavior of BML in our experiments.

There are many types of malware behaviors in our data set. In order to do further classification for different behaviors, combining several trained neural networks to be a cascade classifier might be a possible solution. Breiman [32] combines the prediction of the tree classifiers. Bell and Koren [33] also verify the feasibility of combining multiple predictors. To synthesize the classification results of each neural network to determine the class of a system call sample might be a feasible approach. However, Krizhevsky et al. [3] consider that it appears to be too much cost for complex neural networks. Thus, we found another solution for multi-class classifying, the softmax neural network.

Neural networks which contain softmax transformation in their output layer are suitable for multi-class classifying. These neural networks are usually trained with back-propagation gradient-descent procedure. [34] Lawrence et al. [2] trained convolutional neural networks with softmax function to recognize faces. Krizhevsky et al. [3] build large neural networks with softmax and classify 1,000 types of images. Karpathy et al. [4] also verified the classification ability of softmax neural networks on large scale video data set. Former studies have proved the effectiveness of neural networks with softmax function. We decide to follow the MNIST instruction in TensorFlow [35], applying the softmax neural network to be the multi-class classifier for practical application.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

3 Methodology

3.1 Resistant Learning on Single Hidden Layer Feed-forward Neural Network

SLFN had been proved its ability to representing a complex input-output relationship. [36] Given a set of N observations, $(x^1,\ y^1)$, ..., $(x^N,\ y^N)$, we can train an SLFN to learn the input-output relationship of the observations. The SLFN can be regarded as a fitting function $f(x, w)$, where w is the parameter vector. The value $f(x^c, w)$ is expected to be equal or very close to y^c. To train such a fitting function, we should define the loss function of SLFN. We use one of the most commonly adopted methods, least squares estimator (LSE), to evaluate the loss of the model. The LSE of the c^{th} observation (x^c, y^c) is defined as (1). The ultimate goal of training the SLFN model is to minimize $\sum_{c=1}^{N}(e^c)^2$. Gradient descent is a popular method for optimizing SLFN. We applied gradient descent method to minimize LSE in this paper.

$$(e^c)^2 = (y^c - f(x^c, w))^2 \tag{1}$$

From the perspective of statistics, outliers are the observations that lie far away from the fitting function, i.e., outliers have larger square error $(e^c)^2$. Resistant learning means that the machine learning progress are not significantly influenced by the outliers. Resistant learning has a similar meaning to robust learning, but resistant learning does not need previous knowledge for the majority and the outliers. Resistant learning will select the appropriate majority of observations according to a guideline, then, train the model with the majority, and, apply the special method for learning outliers.

Tsaih and Cheng [7] proposed a resistant learning algorithm using SLFN to find a near-perfect fitting function for all of the numerical observations. Nevertheless, such training process is very time-consuming. Huang et al. [9] proposed the resistant learning procedure with envelope. It eases the restriction of fitting function for numeric observations and is more effective on resistant learning. However, past solutions cannot be appropriately used

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

for nominal data. Therefore, we propose a nominal majority learning mechanism, using the resistant learning procedure to cope with the observations that are difficult to learn, perform robust learning to avoid the interference from outliers.

3.2 Bipartite Majority Learning

The proposed resistant learning method, bipartite majority learning (BML), focuses on binary classification problem, and SLFN is used as the learning model. The SLFN architecture is defined in (2) to (4). Table 1 shows the description of the notations. $a_i(x)$ is the net output value of the hidden layer, and $f(x)$ is the output value of the SLFN. The activation function, $tanh$, is hyperbolic tangent. The SLFN can be a bipartite classifier by setting a threshold [12]. If the output value of an observation is greater or equal than the threshold, it will be considered as class 1; otherwise class 2. Figure 1 shows the tensor graph of SLFN.

Table 1: Table of Notations

Notation	Description
x^c	$x \equiv (x_1, , x_m)^T$, x^c is the c^{th} of input observations.
y^c	The desired output corresponding to the c^{th} input observation of x^c.
m	The dimension of input observation x.
p	The number of adopted hidden nodes.
w_{i0}^H	The bias value θ of the i^{th} hidden node.
w_{ij}^H	The weight between x_j and the i^{th} hidden node.
w_0^O	The bias value θ of the output node.
w_i^O	The weight between the i^{th} hidden node and the output node.

$$f(x) \equiv w_0^O + \sum_{i=1}^{p} w_i^O a_i(x) \tag{2}$$

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

$$a_i(x) \equiv \tanh(w_{i0}^H + \sum_{j=1}^{m} w_{ij}^H x_j) \tag{3}$$

$$\tanh(x) \equiv \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4}$$

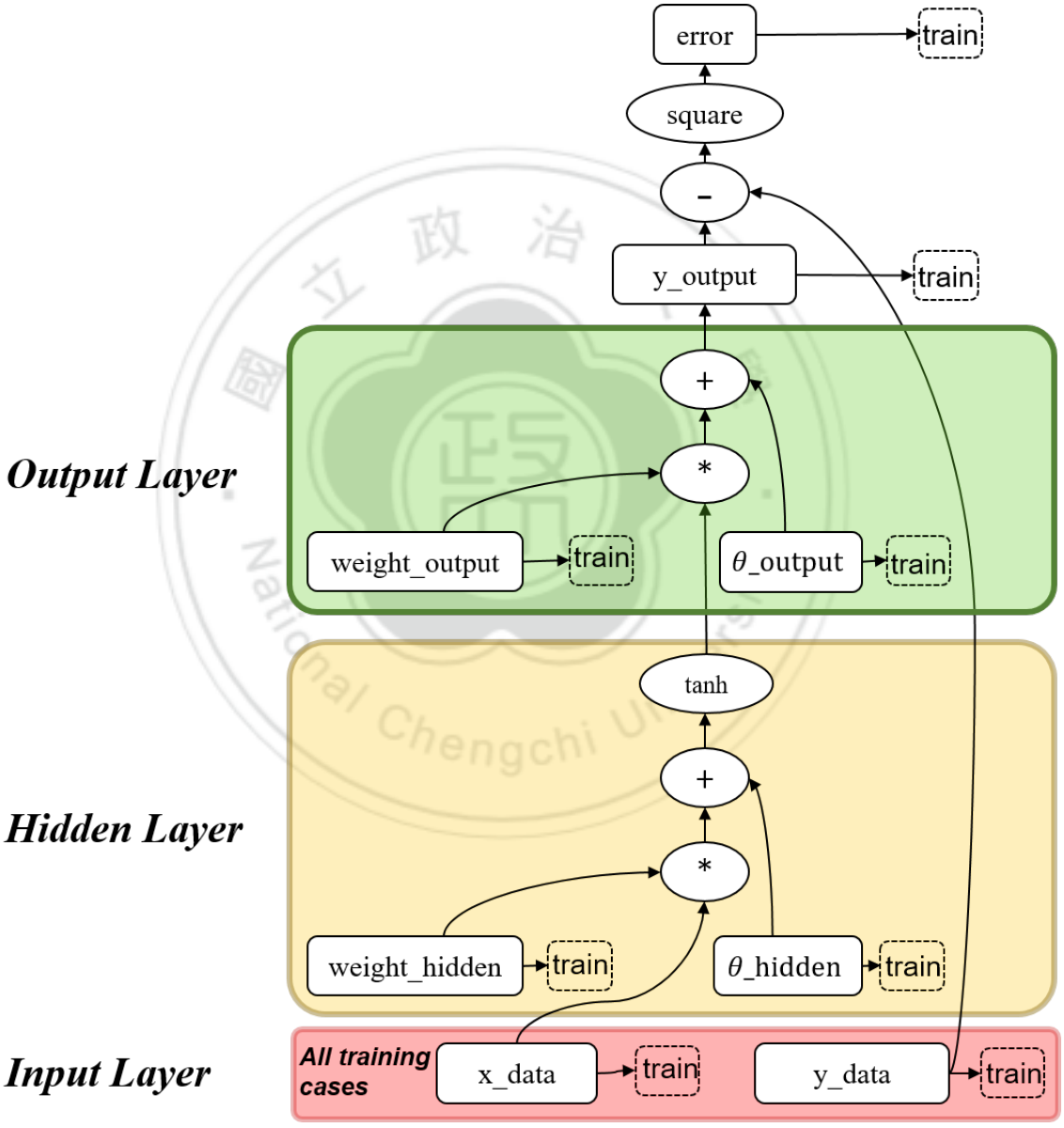


Figure 1: The tensor graph of SLFN.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

Algorithm 1 Define the SLFN tensor graph in python code

1: $x \leftarrow tf.placeholder(tf.float64)$

2: $y \leftarrow tf.placeholder(tf.float64)$

3: $ht \leftarrow tf.Variable(h\_t)$

4: $hw \leftarrow tf.Variable(h\_w)$

5: $hidden\_layer \leftarrow tf.tanh(tf.add(tf.matmul(x, hw), ht))$

6: $ot \leftarrow tf.Variable(o\_t)$

7: $ow \leftarrow tf.Variable(o\_w)$

8: $y\_ \leftarrow tf.add(tf.matmul(hidden\_layer, ow), ot)$

9: $sr \leftarrow tf.reduce\_sum(tf.square(y - y\_))$

10: $train \leftarrow tf.optimizer(eta).minimize(sr)$

Algorithm 1 is the corresponding code in Python. In tensorflow, we should define the calculation relationship between tensors. The binding tensors form a data flow graph, which is the tensor graph(i.e., Figure 1). Variables `x` and `y` are the tensors of `tf.placeholder` type that holds the input data. The variables `hw`(hidden layer weights, w_{ij}^H), `ht`(hidden layer theta, w_{i0}^H), `ow`(output layer weights, w_i^O) and `ot`(output layer theta, w_0^O) are the tensors of `tf.Variable` type which will be modified by the optimizer. The structure of the SLFN is defined by the above tensors and certain tensor operations. The `tf.matmul` performs the matrix multiplication, the `tf.add` performs matrix addition and the `tf.square` squares all the elements in the tensor. The `tf.reduce_sum` calculates the sum of all square residuals. Finally, the optimizer, `tf.optimizer`, applies gradient descent method to modify the variable tensors of neuron weights.

In the supervised learning scenario for binary classification, we should give an appropriate label to our training set. In general, the desired output will be set to $[1, 0]$ and $[0, 1]$ for binary classification when the SLFN have two output nodes. But in this study, our SLFN has only one output node, the desired output of the observations are given dynamically by a specific method. The learning goal of the SLFN is to discern the majority of two classes data. We adopt the linearly separating condition (the condition

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

L) [12] to distinguish two classes of observations. The α in (5) is the minimum output value of the observations in class $1(C_1)$ and the β in (6) is the maximum output value of the observations in class $2(C_2)$. If $\alpha > \beta$, for all $f(x^c) : c \in n$, the condition L in (7) is satisfied. The two classes of observations can be separated by a threshold, $\frac{\alpha+\beta}{2}$. In practice, we label C_1 as {1} and C_2 as {-1} at the beginning of training process.

$$\alpha = \min_{y^c \in C_1} f(x^c) \tag{5}$$

$$\beta = \max_{y^c \in C_2} f(x^c) \tag{6}$$

$$\textit{The Linear Separating Condition} : \alpha > \beta \tag{7}$$

Since we introduce the condition L to be the learning goal of the SLFN, it could be training more faster than the envelope method. The envelope method proposed by Huang et al. [9] ensures that the square error between y^c and $f(x^c)$ should be less than two times of the standard deviation. The condition L is less restrictive than the envelope module but more appropriate to do bipartite classification.

Table 2 presents the proposed bipartite majority learning algorithm. Assume there are N observations, and γ is the majority rate while $\gamma N > m + 1$. The BML algorithm is terminated when more than γN observations are correctly classified and the condition L is satisfied.

Let $S(N)$ be the set of N observations. Let the n^{th} stage be the stage of handling n reference observations (i.e., $S(n)$), and $\gamma N \geq n > m + 1$. Let $\hat{S}(n)$ be the set of the observations which are classified correctly by the condition L at the end of n^{th} stage. Then, the acceptable SLFN estimate that leads to a set of $\{(x^c, y^c)\}$ that can find a threshold to separate the two classes of observations for all $c \in S(n)$. Meanwhile, $|\hat{S}(n)| \geq n$ since $S(n) \subseteq \hat{S}(n)$. To put it another way, at the end of the n^{th} stage, the acceptable SLFN estimate presents a fitting function f can find a threshold to classify at least n

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

observations in $\{(x^c, y^c) : c \in \hat{S}(n)\}$.

Table 2: The bipartite majority learning algorithm

Step 1	Randomly obtain the initial $m+1$ reference observations, two classes of observations each account for half of the m+1 observations. Let $S(m+1)$ be the set of observations of these observations. Set up an acceptable SLFN estimate with one hidden node regarding the reference observations (x^c, y^c) for all $c \in S(m+1)$. Set $n = m+2$.
Step 2	If $n > \gamma N$, STOP.
Step 3	Present the n-1 reference observations (x^c, y^c) that are the ones with the largest distances between C_1 and C_2. Then select another observation (x^k, y^k) so that the value of $\alpha - \beta$ will be the largest. Let $S(n)$ be the set of observations selected in stage n.
Step 4	If n reference observations satisfy the condition L, go to Step 7.
Step 5	Set $\tilde{w} = w$
Step 6	Apply the gradient descent algorithm to adjust weights w until one of the following cases occurs: (1) If n reference observations satisfy L, go to Step 7. (2) If the n observations cannot satisfy L, then restore the weights. Set $w = \tilde{w}$ and apply the resistant learning mechanism by adding extra hidden nodes to obtain an acceptable SLFN.
Step 7	$n+1 \to n$; go to Step 2.

The proposed BML executes the following two procedures: (i) the ordering procedure implemented by Step 3 that determines the input sequence of reference observations and (ii) the modeling procedure implemented by Step 6 that adjusts the weights of the SLFN to minimize the sum of square residuals $\sum_{c=1}^{N}(e^c)^2$. If the gradient descent mechanism

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

cannot tune the weight to find an acceptable SLFN, then restore weights and adjust the number of hidden nodes adopted in the SLFN. Finally, all n observations $S(n)$ at the n^{th} stage would satisfy the condition L. The detail operations are explained as follows.

(Step 1) It first randomly chooses $m+1$ observations from N training data. Then, it calculates the weight of the initial neural network by using the $m+1$ reference observations in the initial training case. The initial weights of the neural network are given by formula (8) to (11). We firstly calculate w_0^O and w_1^O in (8) and (9) by all of the reference observations. Next, we calculate w_{i0}^H and w_{ij}^H in (8). There are $m+1$ hidden weight variables, we can use $m+1$ reference observations to obtain a set of $m+1$ simultaneous equations. Then, we can solve the $m+1$ simultaneous equations by using matrices [37] to get the desired hidden weight values, and make $f(x^c) = y^c \ \forall \ c \in S(m+1)$.

$$w_0^O = \min_{c \in S(N)} y^c - 1 \tag{8}$$

$$w_1^O = \max_{c \in S(N)} y^c - \min_{c \in S(N)} y^c + 2 \tag{9}$$

$$\tilde{y}^c = \tanh^{-1}\left(\frac{y^c - \min_{c \in S(N)} y^c + 1}{\max_{c \in S(N)} y^c - \min_{c \in S(N)} y^c + 2}\right) \tag{10}$$

$$w_{i0}^H + \sum_{j=1}^{m} w_{ij}^H x_j^c = \tilde{y}^c \ \forall \ c \in S(m+1) \tag{11}$$

Algorithm 2 shows how we use the TensorFlow API to define the operations of equations (8) to (11).

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

Algorithm 2 Calculate the first SLFN weights in python code

1: $o\_w \leftarrow tf.max(y) - tf.min(y) + 2$

2: $o\_t \leftarrow tf.min(y) - 1$

3: $s\_y \leftarrow y[: m + 1]$

4: $yc \leftarrow tf.arctanh((s\_y - o\_t)/o\_w)$

5: $s\_x \leftarrow x[: m + 1]$

6: $h\_t\_vector \leftarrow tf.ones([m + 1, 1])$

7: $xc \leftarrow sess.run(tf.concat([s\_x, h\_t\_vector]))$

8: $answer \leftarrow sess.run(tf.matrix\_solve\_ls(xc, yc))$

9: $h\_w \leftarrow answer[: m]$

10: $h\_t \leftarrow answer[m :]$

The purpose of using this method to set the weight is to ensure that the initial neural network has met the condition: $e^c = 0$ for all $c \in S(m + 1)$. That is, the initial SLFN perfectly represents the correspondence between x^c and y^c for the $m + 1$ reference observations.

(Step 2) It is the termination condition of the system. We set the majority rate, γ, to 95%. It guarantees the SLFN can correctly discern the observations in training set more than 95%.

(Step 3) The BML first computes all the possible values of $\alpha - \beta$ of $n - 1$ observations from all N observations. It then selects a set of $n - 1$ observations that has the maximal value of $\alpha - \beta$. To find the $n-1$ observations, we firstly sort the values $f(x^c)$ in C_1 and C_2, respectively. Then we can get i maximum $f(x^c)$ in C_1 and get $(n-1-i)$ minimum $f(x^c)$ in C_2, $i \in [1, n-2]$, to calculate all possible $\alpha - \beta$. The time complexity for obtaining such $n-1$ observations is $O(NlogN)$ since the time complexity of sorting is $O(NlogN)$ and the time complexity of calculating all possible $\alpha - \beta$ is $O(n)$, $N > n$. Compare to the training process, this step does not significantly reduce the efficiency of learning. After selecting the $n - 1$ observations, it picks another observation, (x^k, y^k), so that the value of $\alpha - \beta$ will be the largest. The purpose of this selection mechanism is to select the n observations

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

which most likely to be classified by the condition L. The $(n-1)^{th}$ stage $S(n-1)$ asserts that there is at least one set of $n-1$ observations that can let $\alpha - \beta > 0$. Although the n-1 observations selected in the n^{th} stage may not necessarily equal to $S(n-1)$, this select mechanism ensures that if the observation (x^k, y^k) is excluded, the $n-1$ observations can satisfy the condition L.

(Step 4) It checks if the n selected reference observations satisfy the condition L. If true, the n^{th} stage can find at least one set of n observations that can let $\alpha - \beta > 0$, then it goes to the next stage. If not, the BML would temp to find an acceptable SLFN for the chosen observations $S(n)$.

(Step 5) It saves the current weights of SLFN for the resistant learning procedure. At the end of the $(n-1)^{th}$ stage, the ones in $\{(x^c, y^c) : c \in S(n-1)\}$ satisfy the condition L. The resistant learning procedure can cram a new observation (i.e., (x^k, y^k)) by adding two hidden nodes while not affecting the output of other observations. Adjusting the weights by gradient descent mechanism in Step 6 might make the $n-1$ observations picked in the stage n violating the condition L. Therefore, the current state of the neural network needs to be temporarily stored so that it can be restored in Step 6-2.

(Step 6) We apply gradient descent mechanism to find an acceptable SLFN. For the purpose of nominal supervised learning, the learning target was given dynamically rather than fixed value. Although we respectively give the desired output y^c for C_1 and C_2 observations at the beginning, the difference between the two classes observations is even more important. Let $\bar{S}(n)$ be the subset of $S(n)$, $S(n) = \{k\} + \bar{S}(n)$. We first calculate $\max(f(x_{C_1}))$ and $\min(f(x_{C_2}))$ $\forall c \in \bar{S}(n)$, then the supervised learning target changes to $\max(f(x_{C_1}))$ $\forall (x^c, y^c) \in C_1$, and the supervised learning target changes to $\min(f(x_{C_2}))$ $\forall (x^c, y^c) \in C_2$. After setting the learning target of $\bar{S}(n)$, we compute the values α and β of $\bar{S}(n)$. Then, if $y^k \in C_1$, the learning target of x^k is set to α; otherwise $y^k \in C_2$, the learning target of x^k is set to β. Then, the gradient descent mechanism is applied to adjust the weights to find an acceptable SLFN.

(Step 6.1) If an acceptable SLFN is found, we move to the next stage. However, we

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

might encounter the problem of local optimum which is caused by the implementation of the gradient descent mechanism or by the SLFN model that does not have enough hidden nodes. Both situations lead to an unacceptable SLFN estimate regarding the n reference observations. Therefore, we adopt the resistant procedure proposed by Tsaih and Cheng [7] to cope with the observation x^k.

(Step 6.2) We apply the resistant learning procedure. Restore the $\tilde{w}$ that is stored in Step 5. Then, we add two hidden nodes to change the output value of x^k to the learning target. It can also represent the observation x^k is closer to the threshold than other same class observations. Other output values $y^{c\prime}\ \forall\ c \in \bar{S}(n)$ will not be significantly affected by the resistant learning procedure. The hidden weights formulas of newly hidden nodes are defined in (12) to (16).

$$w^H_{p-1,0} = \zeta - \lambda\alpha^T x^k \tag{12}$$

$$w^H_{p,0} = \zeta + \lambda\alpha^T x^k \tag{13}$$

$$w^H_{p-1} = \lambda\alpha^T \tag{14}$$

$$w^H_{p} = -\lambda\alpha^T \tag{15}$$

$$w^O_{p-1} = w^O_p = \frac{|y^{k\prime} - w^O_0 - \sum_{i=1}^{q}(w^O_i w^k_i)|}{2\tanh(\zeta)} \tag{16}$$

ζ is a small constant number set to 0.05. λ is a large constant number set to 10^5. α^T is an m-dimension vector which length equals 1 and satisfies the condition (17).

$$\alpha^T(x^k - x^c) \neq 0\ \forall\ c \in I(n) - \{k\} \tag{17}$$

By adding two hidden nodes in the hidden layer, the SLFN satisfies the condition L

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

for the n reference observations $S(n)$. The output value $f(x^k)$ will be very close to α if $y^k \in C_1$; otherwise $y^k \in C_2$, the output value $f(x^k)$ will be near to β.

(Step 7) We increase n by 1 and repeat step 2 to step 7. Most machine learning methods use all training data as the basis for adjusting weights. In the BML mechanism, in order to avoid anomaly observations affecting SLFN learning and picking appropriate majority observations, BML will start with a small amount of data and gradually increase the amount of selected data. The advantage of this approach is that since we do not necessarily know in advance which data is the majority and which data is the anomaly in all training data, it is possible that anomaly data will be selected when $m+1$ data are first acquired. However, as the number of selected data n increases, the selection mechanism in step 3 will select those that are most easily classified by condition L. Since n observations selected in the n^{th} stage is most suitable for the current SLFN, the n observations do not necessarily include the $n-1$ observations selected in the $(n-1)^{th}$ stage. Through this dynamic selection method, we can select appropriate majority data and avoid the anomaly's impact on the effectiveness of learning. Although n only increases by 1 at one time that would slower than training the SLFN with all data, we found that the SLFN does not need to retrain at every stage in our experiments due to the observations can satisfy the condition L. The BML mechanism can quickly move to the next stage when the most part of training data can be classified correctly.

3.3 Majority Learning on Softmax Neural Network

The softmax neural network is a popular neural network architecture, which is commonly adopted to do nominal classification. In general conditions, the softmax neural network will learn for all training data, but in order to make softmax neural network a comparable learning model benchmark, we design a majority learning method for softmax neural network.

The designed softmax majority learning method has 6 procedures which are similar to the proposed majority method, BML. We list the process of softmax majority learning

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

in table 3.

Table 3: The softmax majority learning algorithm

Step 1	Randomly obtain the initial $m+1$ reference observations, two classes of observations each account for half of the m+1 observations. Let $S(m+1)$ be the set of observations of these observations. Set up an acceptable SLFN estimate regarding the reference observations (x^c, y^c) for all $c \in S(m+1)$. Set $n = m+2$.
Step 2	If $n > \gamma N$, STOP.
Step 3	Present the n reference observations (x^c, y^c) that are the ones with the smallest cross entropy value. Let $S(n)$ be the set of observations selected in stage n.
Step 4	If n reference observations can be correctly classified by the model, go to Step 6.
Step 5	Apply the gradient descent algorithm to adjust weights w until one of the following cases occurs: (1) If n reference observations can be correctly classified by model, go to Step 6. (2) If the n observations stuck in local optimal and cannot correctly classified by model, go to step 6.
Step 6	$n+1 \rightarrow n$; go to Step 2.

The designed softmax majority learning method has the same procedure in increasing training data and terminal condition. That is to say, compare to the general training method of the softmax neural network, we iteratively obtain a new observation to be the training data in each stage. We modify the other steps to let the softmax neural network fit the majority learning process. In step 1, we arbitrarily pick $m+1$ observations to be the initial training data. We adopt gradient descent method to get an initial network state that the $m+1$ observations can correctly classified by the softmax neural network. Note that the gradient descent method cannot always tune the weight to find an acceptable neural network, we can only minimize the cross-entropy of the model. In step 2, we

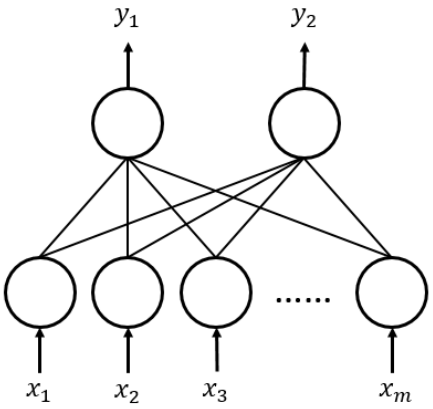
DOI:10.6814/THE.NCCU.MIS.001.2019.A05

check the terminal condition just like BML. In step 3, we do forward pass for all N observations, calculate each observations' cross-entropies, and sort the N observations by the cross-entropy values. We obtain n observations with the smallest cross-entropy values to be the training data in the satge n. In step 4, we would check if the n observations with the smallest cross-entropy values can correctly classified by the softmax neural network. If yes, go to step 6. Otherwise, go to step 5, the softmax neural network should apply gradient descent optimizer to tune the weights until one of the two following occurs: (1) The softmax neural network can correctly classified the n observations. (2) The gradient descent optimizer stuck in local optimal and could not find a set of weight correctly classified the n observations.

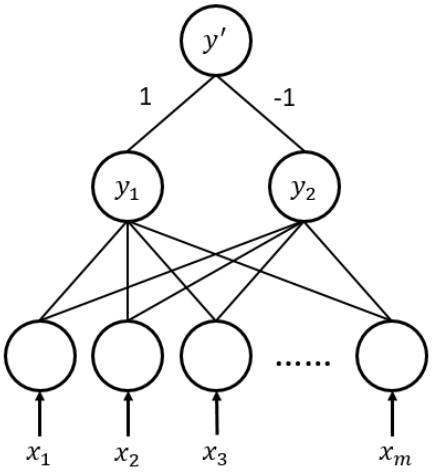
The softmax majority learning method does not have a resistant learning procedure. After the training process, the ANN model cannot make sure the γN majority can be correctly classified. In other words, the softmax majority learning method has a limited ability to deal with outliers.

The ANN model with softmax function in softmax majority learning method could be regarded as another form of condition L. Because of we focus on bipartite classification problem, the ANN model have two output nodes in the output layer. If we concatenate a new layer with one node to the original two output nodes and the neuron weights connected are 1 and -1 respectively, the newly output node would satisfy the condition L. One of the two class would have output value greater than 0, the other class would have output value less than 0. Figure 2 illustrates the effect of this operation.

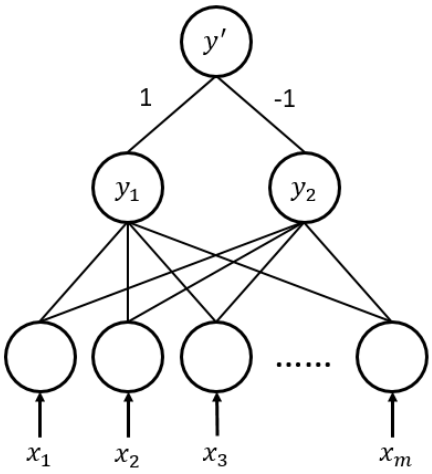
DOI:10.6814/THE.NCCU.MIS.001.2019.A05



(i) Origin ANN architecture in softmax majority learning



(ii) Case: Input data belongs to class 1. $y_1 > y_2$, so $y' > 0$.



(iii) Case: Input data belongs to class 2. $y_1 < y_2$, so $y' < 0$.

Figure 2: The condition L in another form.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

3.4 Majority Learning on Support Vector Machines

Support Vector Machines (SVMs) are a prevalent supervised learning method for data classification, regression, and other learning tasks. [38] SVMs have several advantages, for example, SVMs are effective in high dimensional spaces and still effective in cases where number of dimensions is greater than the number of samples. SVMs have several kernel functions that can be specified for the decision function. We also design a majority learning method for SVM. Table 4 describe the SVM majority learning process.

Table 4: The SVM majority learning algorithm

Step 1	Randomly obtain the initial $m+1$ reference observations, two classes of observations each account for half of the m+1 observations. Let $S(m+1)$ be the set of observations of these observations. Set up an initial model regarding the reference observations (x^c, y^c) for all $c \in S(m+1)$. Set $n = m+2$.
Step 2	If $n > \gamma N$, STOP.
Step 3	Present the n reference observations (x^c, y^c) that are the ones with the smallest $-y^c * f(x^c)$ value. Let $S(n)$ be the set of observations selected in stage n.
Step 4	If n reference observations can be correctly classified by the model, go to Step 6.
Step 5	Apply the weight tuning method to adjust weights w until one of the following cases occurs: (1) If n reference observations can be correctly classified by model, go to Step 6. (2) If the n observations stuck in local optimal and cannot correctly classified by model, go to step 6.
Step 6	$n+1 \rightarrow n$; go to Step 2.

The SVM majority learning algorithm has the same sort-and-select procedure to pick the data most familiar to the trained classifier. The main idea of SVM is to find a

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

hyperplane which can separate two class of data in a high dimension space. Thus, in step 3, we pick the data with the smallest $-y^c * f(x^c)$ value. y^c is the desired output of the training data. y^c is 1 for all class 1 data and y^c is -1 for all class 2 data. $f(x^c)$ is the predict output value of SVM classifier for x^c. If the two values are the same sign, it means that x^c is separated into the correct space by the hyperplane. If the two values are the opposite sign, it means that x^c is separated into the wrong space by the hyperplane. If the value $|f(x^c)|$ is larger, the farther away b is from the hyperplane. Depending on the selection criteria, data that is far from the hyperplane will be prioritized selected.

SVM classification methods have several kernels. Linear, polynomial, radial basis function (RBF) and sigmoid are common used kernels. Each kernel has different ability for classifying different data distribution. Figure 3 illustrates the ability of different kernels classifying specific data distribution.

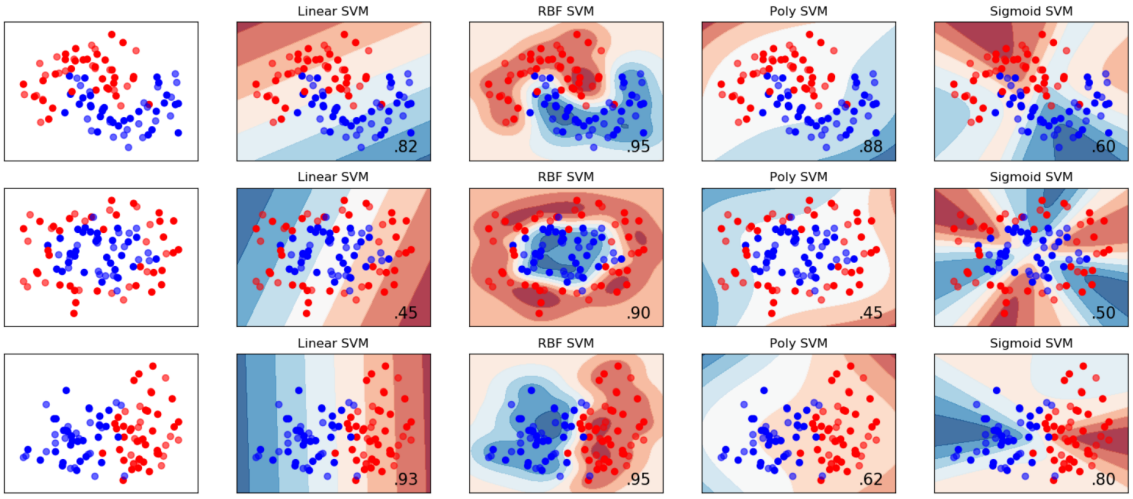


Figure 3: An example for different SVM kernels classifying two class data.

As for the implementation, Scikit-Learn [39] provides high-level functions for building SVMs classifiers. We use Scikit-Learn library for the implementation of SVM majority learning algorithm.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

3.5 Multi-Class Classifier for Majority

For the bipartite classification problem, the obtained SLFN from BML can be used as a bipartite classifier. An input can be classified by comparing its output value with a threshold. The threshold of the SLFN is $\frac{\alpha+\beta}{2}$ where α is the minimum output value in C_1 and β is the maximum output value in C_2 of the majority in the training set. If $f(x^t) \geq \frac{\alpha+\beta}{2}$, x^t is classified as C_1; otherwise C_2.

As for multi-class classification problem, for example, classifying several types of malware behaviors, We need to add an additional mechanism in our neural network. In the literature, a cascade classifier is an option [40]; however, cascade classifier may lead to poor classification accuracy in the circumstance of too many classes. Hence, inspired by the MNIST experiments in TensorFlow [35], we build a neural network with softmax function to perform multi-class classification. This neural network does not contain any hidden layer. The softmax function is a generalization of the logistic function that converts a K-dimensional vector z of arbitrary real values to a K-dimensional vector $\sigma(z)$ of real values in the range [0, 1] that add up to 1. The neural network architecture is defined in (18). We use cross-entropy to determine the loss of our model, and the cross-entropy is defined in (19).

$$g(x^c) \equiv softmax(Wx^c + b) \tag{18}$$

$$H_{y^{c\prime}}(y) = -\sum_{c=1}^{N} y^{c\prime} \log(y^c) \tag{19}$$

Figure 4 shows the tensor graph of the softmax neuron network. Algorithm 3 is the corresponding code in Python. Variables `x` and `y` are the tensors of `tf.placeholder` type that holds the input data. `W` and `b` are the tensors of `tf.Variable` type. The shape of `W` is determined by the dimension of `x` and the dimension of `y`. The `tf.matmul` performs the matrix multiplication, the `tf.add` performs matrix addition. The `tf.reduce_sum` sums up all of the cross entropies.Finally, the optimizer, `tf.optimizer`, applies gradient

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

descent method to modify the variable tensors of neuron weights.

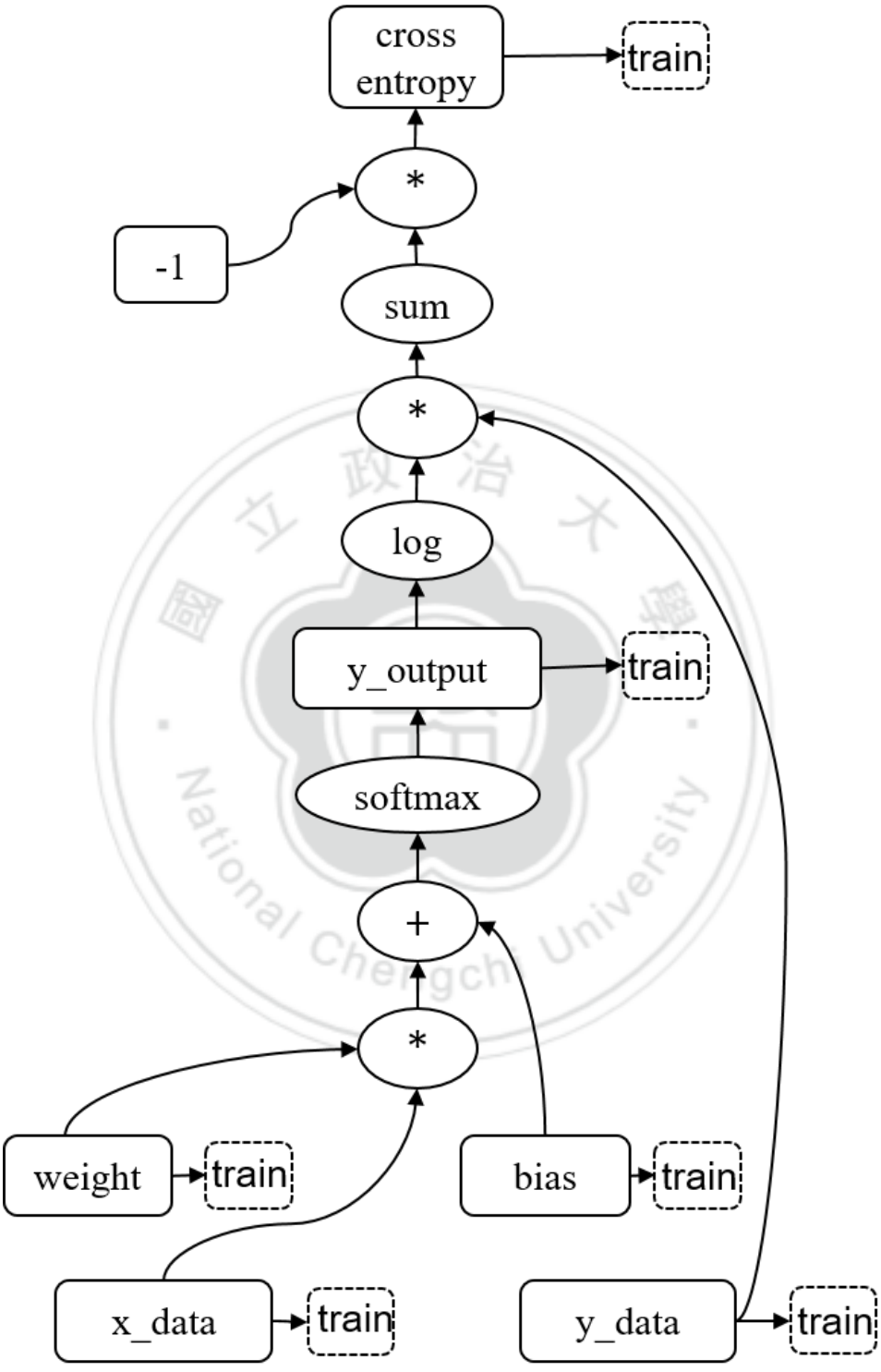


Figure 4: The tensor graph of the softmax neuron network.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

Algorithm 3 Define the Softmax tensor graph in python code

1: $x \leftarrow tf.placeholder(tf.float64)$

2: $y \leftarrow tf.placeholder(tf.float64)$

3: $W \leftarrow tf.Variable([x.shape[1], y.shape[1]])$

4: $b \leftarrow tf.Variable([y.shape[1]])$

5: $y\_ \leftarrow tf.nn.softmax(tf.matmul(x, W) + b)$

6: $c\_e \leftarrow -tf.reduce\_sum(y * tf.log(y\_))$

7: $train \leftarrow tf.optimizer(c\_e).minimize(sr)$

Compared to SLFN, this neural network architecture has neither hidden layer nor activation function. Gradient descent is also adopted here to adjust the weight and bias. The advantage of this neural network is that it can classify multiple malicious behaviors. The aforementioned SLFN can only distinguish between malicious behavior and benign behavior. Using the softmax neural network, we can discern the category of a malicious behavior. The neural network will output a set of probability values representing the likelihood of each category. Among them, we would take the maximum probability value to be the classification result of the neural network.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

4 EXPERIMENTS

In the experiments, we perform dynamic malware analysis and evaluate the effectiveness of the proposed BML on real-world data sets. We apply BML to learn two types of features, malware behavior, and benign program behavior.

4.1 Malware Samples from OWL

In our experiments, 998 malicious programs and 2 benign programs are used. The malware executables are downloaded from the OWL database [41]. Chiu et al. [31] executed the malware samples in a virtual machine with Cuckoo Sandbox [42] installed and used the tool *Viso* [43] to record the chronological data of the system calls. The two benign program are Google Chrome and Filezilla. Then, the chronological data are split into multiple individual samples with the same window size N. Fig. 5 illustrates the method of collecting training data.

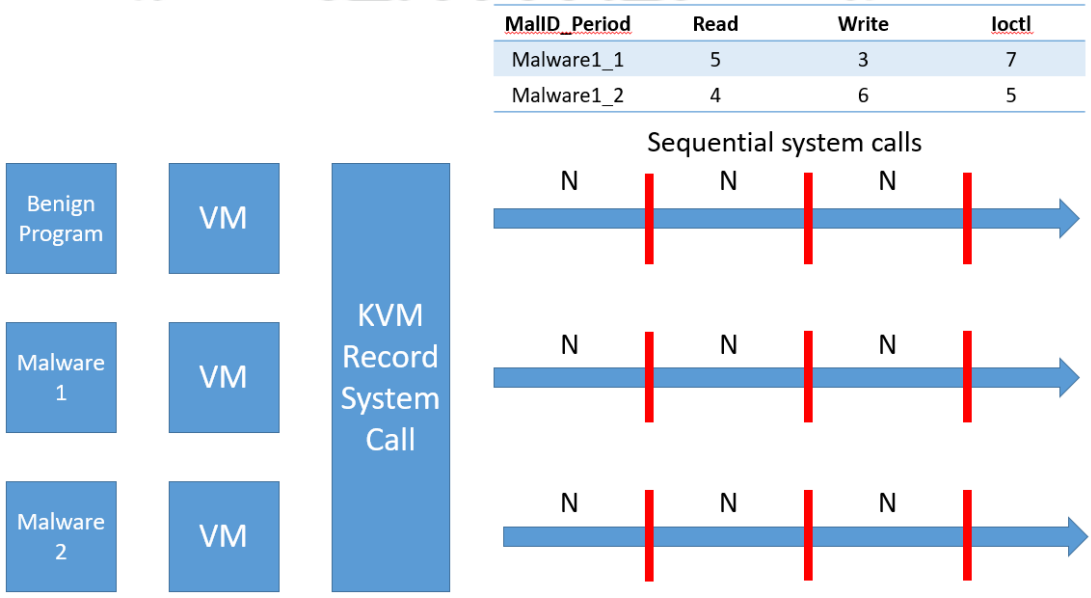


Figure 5: Generate system call samples.

With the window size $N = 1{,}000$, there are 1,229,634 samples in total. The number of distinct system call recorded in these programs is 52. Hence, we can build a vector with 52 dimensions for each sample, and each dimension represents the frequency occurrence of a

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

specific system call. After clustered by the GHSOM algorithm [31], there were 19 clusters containing only malicious behavior. In this case, the rest of the samples that are not in the 19 clusters are considered as non-malicious samples, i.e., benign samples. Hence, the unique patterns of these 19 clusters are the detection rules of malicious behaviors.

Some clusters have many repetitive features of samples. In order to prevent the bias of model training, we filtered out the samples with the same features in advance, and only one sample was reserved for an individual feature. In addition, we need a large enough sample size to be suitable for the division of the training/testing set. Therefore, we select clusters with more than 1000 samples. After these two filtering processes, only 9 malware detection rule clusters meet our need. For the purpose of balancing the amount of benign and malware samples, we randomly sampled 19391 benign samples from 1,187,842 benign samples. The number 19,391 is equal to the sum of 9 malware sample amounts. The sample amount of each detection rules are shown in Table 5.

Table 5: Sample amount in each detection rules

Rule 1.	3175	Rule 2.	1331	Rule 3.	2025	Rule 4.	1838	Rule 5.	2451
Rule 6.	4356	Rule 7.	1208	Rule 8.	1220	Rule 9.	1787	Benign	19391

4.2 Evaluation

To evaluate our model, we compare our BML with the state-of-art methods, including the former resistant learning approach, envelope (ENV) [9], softmax and SVM. In order to maintain the same benchmark, softmax and SVM methods are adapted to majority learning to make a fair comparison.

ENV already has a majority picking method but does not have a classification method for bipartite data. Thus, we combined ENV with the condition L for evaluating ENV's classification accuracy. The ANN with softmax function does not have a majority mothod. So, we designed a majority learning method for softmax neural network in section 3.3. SVM methods do not have majority methods either, so we designed majority learning methods for SVMs in section 3.4. We applied these majority learning method in experi-

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

ment 1.1, 2.1 and 3 to compare the performance.

The mentioned majority learning mechanisms in this study are applied to learn the majority pattern in each cluster. We test all algorithms in the same hardware environment. We implement the majority learning methods with TensorFlow API and use GPU accelerate training efficiency. In this paper, we set γ to 95%, which means all of the majority learning mechanisms stop when the 95% of the majority in the training set is correctly classified. We have tested several different degrees of γ setting. Since our system call data set has been classified by GHSOM, there is a certain difference between the benign sample patterns and the malicious sample patterns. The SLFN can easily separate most of the data. Therefore, we set a stricter majority rate, 95%, to ensure that the SLFN can learn most of the training data patterns and does not interfere with few abnormal patterns.

We test the performance of BML under different experimental designs. In Experiment 1, we focus on the performance of BML to classify the samples in each cluster and test whether BML selecting the appropriate majority. In Experiment 2, we concern about the performance of BML to classify a large number of samples in each cluster and test whether BML selecting the appropriate majority. In Experiment 3, we test if BML has ability classifying two categories of samples which contain a variety of benign and malware patterns.

4.2.1 Exp. 1.1: Majority Learning on Small-Size Sampling Data

RQ 1: How BML performs in terms of efficiency and accuracy compared to the state-of-the-art approaches on small-size data sets?

To answer RQ1, in this experiment, we tested the learning results of BML, ENV, SVM, and softmax on sampling data, whether it can improve performance in the case of majority learning. The performance evaluation shows the classification accuracy and the time efficiency of each the methods.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

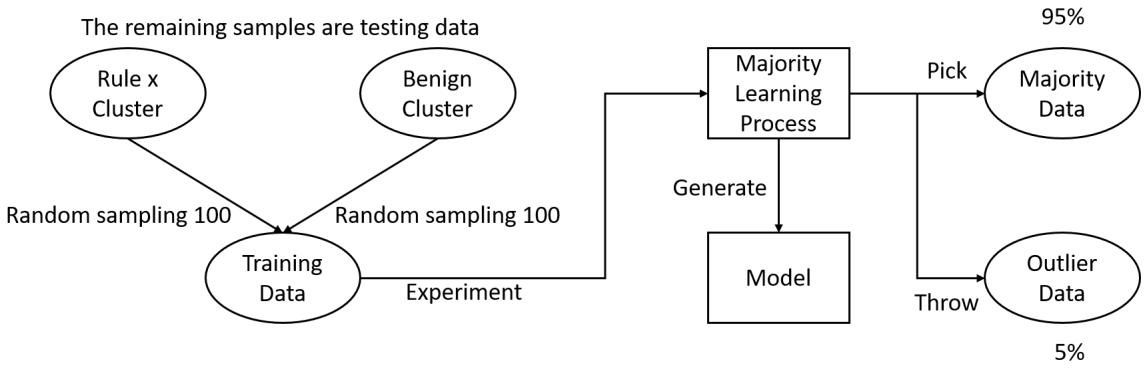


Figure 6: Experimental design of experiment 1.1.

Figure 6 illustrates the experimental design of experiment 1.1. We randomly selected an equal amount of malicious samples from the 9 malware clusters and the benign samples from the benign data set. We decided to select 100 samples from each detection rule clusters, because the size 100 is relatively small compared to the total sample amount in each clusters (less than 10%). These randomly selected samples are used as the training set; the rest of the samples are used as the testing set.

For example, we would randomly select 100 malware samples from the rule cluster 1 and randomly select 100 benign samples from the benign cluster. Then, we labeled these 200 samples according to their class and regarded these samples as training set. The rest of the rule 1 cluster samples and the rest of the benign cluster samples would be the testing set. After we sampling the data from each clusters, we conduct the different majority learning experiments with the same sampling data.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

Table 6: Training result by using 100*2 samples

Rule	Majority Method	#HN	Outliers (#FB/#B, #FM/#M)	Execute Time(s)	Train B #FB/#B	Train M #FM/#M	Test B #FB/#B	Test M #FM/#M
1	SVM_linear	-	(0/1, 0/9)	0.1	0/99	0/91	85/19291	3/3075
	SVM_poly	-	(0/0, 0/10)	0.1	0/100	0/90	127/19291	30/3075
	SVM_rbf	-	(0/0, 3/10)	0.3	0/100	0/90	58/19291	350/3075
	Softmax	-	(0/7, 0/3)	0.6	0/93	0/97	345/19291	1/3075
	ENV	7	(7/9, 0/1)	106.1	0/91	0/99	1078/19291	0/3075
	BML	1	(4/6, 3/4)	0.4	0/94	0/96	759/19291	30/3075
2	SVM_linear	-	(0/6, 0/4)	0.1	0/94	0/96	12/19291	0/1231
	SVM_poly	-	(0/7, 0/3)	0.1	0/93	0/97	18/19291	0/1231
	SVM_rbf	-	(0/7, 3/3)	0.2	0/93	0/97	0/19291	47/1231
	Softmax	-	(0/0, 0/10)	0.5	0/100	0/90	6/19291	7/1231
	ENV	7	(0/0, 0/10)	82.5	0/100	0/90	2/19291	17/1231
	BML	1	(0/4, 0/6)	0.4	0/96	0/94	2/19291	6/1231
3	SVM_linear	-	(1/6, 0/4)	0.1	0/94	0/96	14/19291	0/1925
	SVM_poly	-	(1/7, 0/3)	0.1	0/93	0/97	18/19291	0/1925
	SVM_rbf	-	(0/3, 2/7)	0.2	0/97	0/93	0/19291	48/1925
	Softmax	-	(0/0, 0/10)	0.5	0/100	0/90	3/19291	0/1925
	ENV	5	(0/4, 0/6)	34.5	0/96	0/94	8/19291	0/1925
	BML	1	(0/7, 0/3)	0.4	0/93	0/97	8/19291	0/1925
4	SVM_linear	-	(0/10, 0/0)	0.1	0/100	0/100	15/19291	0/1738
	SVM_poly	-	(1/10, 0/0)	0.1	0/90	0/100	18/19291	0/1738
	SVM_rbf	-	(0/1, 1/9)	0.2	0/99	0/91	0/19291	28/1738
	Softmax	-	(4/10, 0/0)	0.4	0/90	0/100	548/19291	0/1738
	ENV	11	(0/2, 4/8)	104.7	0/98	0/92	0/19291	98/1738
	BML	1	(0/6, 0/4)	0.5	0/94	0/96	0/19291	0/1738
5	SVM_linear	-	(0/4, 0/6)	0.1	0/96	0/94	12/19291	0/2351
	SVM_poly	-	(0/4, 0/6)	0.1	0/96	0/94	18/19291	0/2351
	SVM_rbf	-	(0/0, 4/10)	0.2	0/100	0/90	0/19291	74/2351
	Softmax	-	(7/8, 0/2)	0.5	0/92	0/98	1784/19291	0/2351
	ENV	55	(1/2, 0/8)	1265.0	0/98	0/92	500/19291	0/2351
	BML	1	(0/3, 0/7)	0.4	0/97	0/93	3/19291	0/2351
6	SVM_linear	-	(0/8, 0/2)	0.1	0/92	0/98	16/19291	0/4256
	SVM_poly	-	(0/8, 0/2)	0.1	0/92	0/98	19/19291	0/4256
	SVM_rbf	-	(0/0, 7/10)	0.2	0/100	0/90	0/19291	125/4256
	Softmax	-	(4/10, 0/0)	0.3	0/90	0/100	814/19291	0/4256
	ENV	39	(0/3, 0/7)	461.0	0/97	0/93	0/19291	0/4256
	BML	1	(0/3, 0/7)	0.4	0/97	0/93	0/19291	0/4256
7	SVM_linear	-	(0/4, 0/6)	0.1	0/96	0/94	17/19291	0/1108
	SVM_poly	-	(0/4, 0/6)	0.1	0/96	0/94	34/19291	0/1108
	SVM_rbf	-	(0/0, 9/10)	0.3	0/100	0/90	0/19291	171/1108
	Softmax	-	(0/0, 5/10)	0.5	0/100	0/90	1/19291	79/1108
	ENV	25	(0/0, 4/10)	378.3	0/100	0/90	0/19291	38/1108
	BML	1	(0/0, 4/10)	0.4	0/100	0/90	0/19291	36/1108
8	SVM_linear	-	(0/0, 0/10)	0.1	0/100	0/90	10/19291	0/1120
	SVM_poly	-	(0/0, 0/10)	0.1	0/100	0/90	16/19291	0/1120
	SVM_rbf	-	(0/1, 3/9)	0.2	0/99	0/91	0/19291	58/1120
	Softmax	-	(0/4, 0/6)	0.4	0/96	0/94	35/19291	0/1120
	ENV	37	(0/1, 0/9)	1472.5	0/99	0/91	44/19291	32/1120
	BML	1	(0/6, 0/4)	0.4	0/94	0/96	11/19291	0/1120
9	SVM_linear	-	(0/2, 0/8)	0.1	0/98	0/92	11/19291	0/1687
	SVM_poly	-	(0/1, 0/9)	0.1	0/99	0/91	16/19291	0/1687
	SVM_rbf	-	(0/0, 9/10)	0.2	0/100	0/90	0/19291	60/1687
	Softmax	-	(2/10, 0/0)	0.4	0/90	0/100	664/19291	0/1687
	ENV	5	(0/1, 0/9)	92.1	0/99	0/91	118/19291	0/1687
	BML	1	(0/1, 0/9)	0.4	0/99	0/91	7/19291	0/1687

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

Table 6 shows the training result of SVM, softmax, ENV and BML. For each cluster, we train the different models by using randomly selected samples as training data (100 benign samples and 100 malicious samples are used) in this experiment. The column of "#HN" specifies the number of hidden nodes in SLFN after trained by ENV and BML. Note that the softmax neural network does not have hidden layer, so the amount of hidden nodes must be always zero.

For the ENV, all of the rules need more than one hidden nodes to find the fitting function. For BML, all rules only need one hidden node to classify the majority data. It indicates we do not even need to apply the add hidden nodes procedure to deal with the outliers in the training data.

In column "Outliers (#FB/#B,#FM/#M)", the value #B is the number of benign samples which were regarded as outliers in the training data. The value #M is the number of malware samples which were regarded as outliers in the training data. The sum of #B and #M is equal to 5% of training data because our majority rate is set to 95%. The value #FB and #FM are the number of false classified samples in the benign and malware outliers, respectively. Although outliers have a greater loss than the majority data, not all outliers are misclassified. Because we applied the condition L for classification, if the losses are not great enough, the outliers would not be misclassified by the model.

On the average, BML has higher classification accuracy on training data than EVN, and most of the misclassified samples are benign samples. As for the training time, BML is outperformed then ENV and is similar with SVM and softmax, since BML do not need to re-train the model as many times as ENV.

In this study, we evaluate the accuracy of the model by "false rate". We define the false rate as follows: $False\ Rate = False\ classified\ sample\ amount\ /\ Total\ sample\ amount$. For example, if a rule 1 sample was classified as benign sample by a model, the rule 1 sample is a false classified sample. We sum the amount of false classified rule 1 samples and divide by the total amount of rule 1 samples to calculate the false rate of rule 1 samples. This calculation method applies to all rule clusters and benign clusters.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

In column "Train B(#FB/#B)" indicates the false rate of benign training data, the value #B is the number of benign samples in the training data. In column "Train M(#FM/#M)" indicates the false rate of malware training data, the value #M is the number of malware samples in the training data. In column "Test B(#FB/#B)" indicates the false rate of benign testing data, the value #B is the number of benign samples in the testing data. In column "Test M(#FM/#M)" indicates the false rate of malware testing data, the value #M is the number of malware samples in the testing data.

Figure 7: False rate of different majority learning methods on training data (100*2 samples).

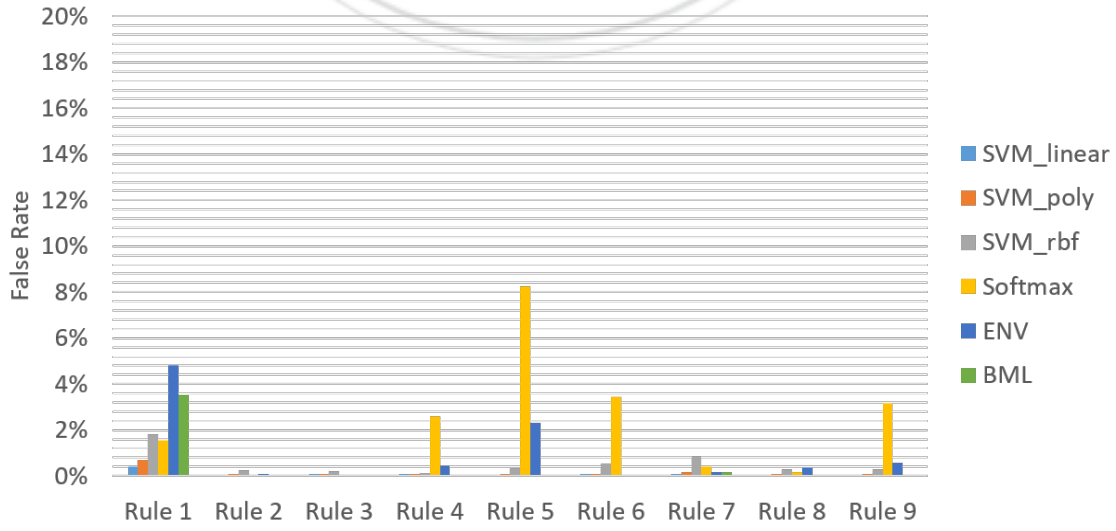


Figure 8: False rate of different majority learning methods on testing data (100*2 samples).

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

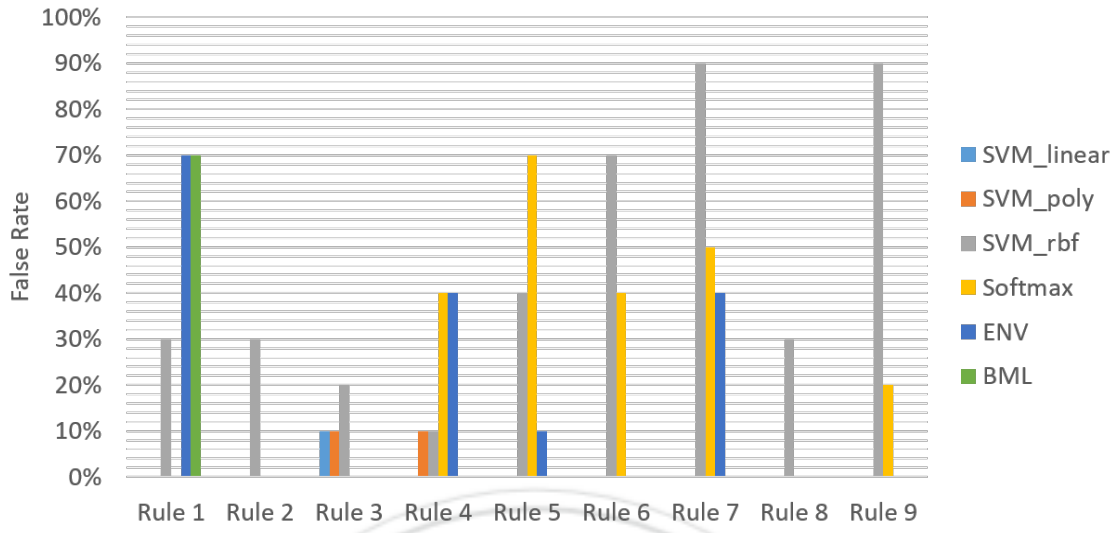


Figure 9: False rate of different majority learning methods on outlier data (100*2 samples).

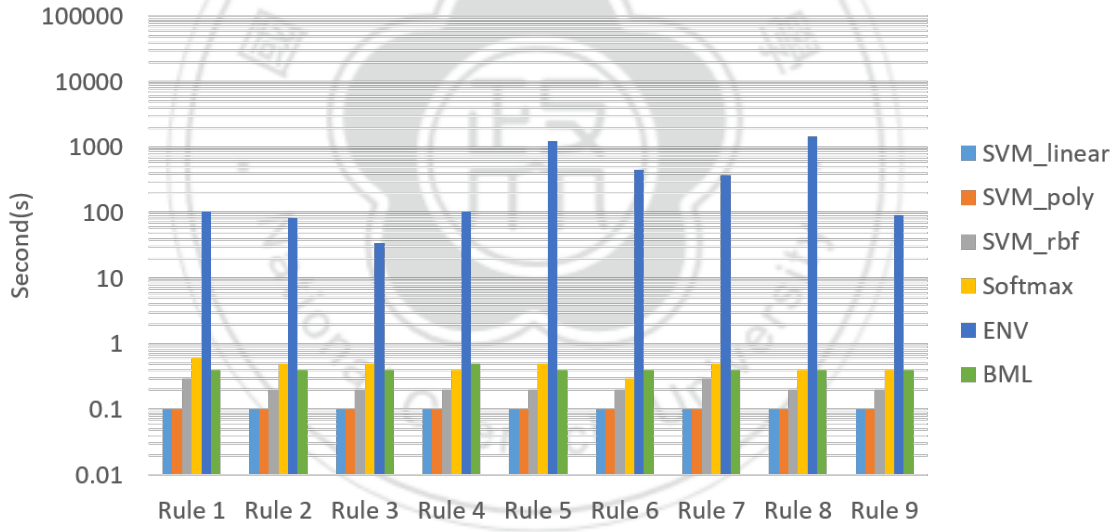


Figure 10: Execution time of different majority learning methods (100*2 samples).

Figure 7 to Figure 9 show the false rate of SVM, softmax, ENV and BML. We calculate the mean false rate of 9 rules, BML can perform higher classification accuracy compare to softmax and ENV on testing data. As for the training data, BML has higher classification accuracy on benign data but has lower classification accuracy than softmax on malware data. Figure 10 shows the execution time of SVM, softmax, ENV and BML. BML, SVM and softmax finish the model training process much faster than ENV.

To answer RQ1, BML has on average higher time efficiency and higher classification

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

accuracy than the state-of-art methods on small size data sets.

4.2.2 Exp. 1.2: Use ANN to Learn the Majority

RQ 2: How well the majority set selected by BML represents on small-size data sets?

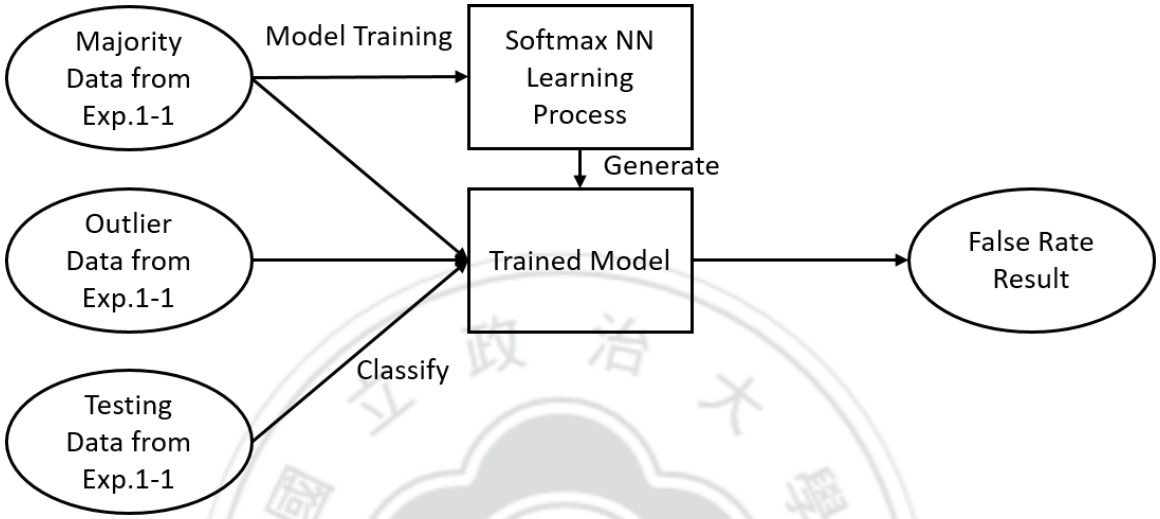


Figure 11: Experimental design of experiment 1.2.

Figure 11 illustrates the experimental design of experiment 1.2. As far as we know, outliers would affect the learning of softmax neural network. If training data contains outliers, the classification accuracy of training and testing data would be decrease. In this experiment, we want to test if BML can select proper majority data by the selecting mechanism. We adopt the softmax neural network to the majority selected by the different majority learning methods. We also test softmax learning by directly using the original training data. We tested whether the selected majority can increase the performance of softmax neural network learning. If the majority are chosen properly, the softmax neural network should learn more accurately from the training data.

We label the data by using the corresponding one-hot vector. A one-hot vector is a vector that in a single dimension is 1 and in other dimensions are 0s. In our case, the data in the n^{th} rule is labeled with a vector that its n^{th} dimension is 1 and other dimensions are 0s. For a benign sample, only the 0^{th} dimension is 1 and others are 0s. Hence, the shape of the one-hot vector is (10, 1). We apply gradient descent 10,000 times for weight tuning and compare the classification accuracy. The learning rate is set as 0.0001.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

Table 7: Softmax neural network classification result by using 100*2 samples

Rule	Majority Method	Outlier FR	Train FR	Test FR
1	None	-	0/100	13/3075
	SVM_linear	0/9	0/91	17/3075
	SVM_poly	0/10	0/90	19/3075
	SVM_rbf	0/10	0/90	17/3075
	Softmax	0/3	0/97	16/3075
	ENV	0/1	0/99	17/3075
	BML	0/4	0/96	19/3075
2	None	-	2/100	34/1231
	SVM_linear	0/4	2/96	42/1231
	SVM_poly	0/3	2/97	41/1231
	SVM_rbf	1/3	1/97	30/1231
	Softmax	5/10	1/90	75/1231
	ENV	1/10	1/90	42/1231
	BML	1/6	2/94	43/1231
3	None	-	14/100	150/1925
	SVM_linear	1/4	7/96	128/1925
	SVM_poly	1/3	8/97	131/1925
	SVM_rbf	0/7	14/93	149/1925
	Softmax	1/10	13/90	158/1925
	ENV	1/6	11/94	148/1925
	BML	0/3	9/97	133/1925
4	None	-	9/100	199/1738
	SVM_linear	0/0	8/100	181/1738
	SVM_poly	0/0	8/100	184/1738
	SVM_rbf	2/9	8/91	220/1738
	Softmax	0/0	7/100	164/1738
	ENV	0/8	8/92	182/1738
	BML	0/4	8/96	180/1738
5	None	-	2/100	207/2351
	SVM_linear	2/6	4/94	246/2351
	SVM_poly	2/6	4/94	247/2351
	SVM_rbf	1/10	2/90	195/2351
	Softmax	0/2	3/98	216/2351
	ENV	2/8	2/92	234/2351
	BML	3/3	3/97	233/2351

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

Table 8: Softmax neural network classification result by using 100*2 samples (cont.)

Rule	Majority Method	Outlier FR	Train FR	Test FR
6	None	-	22/100	922/4256
	SVM_linear	1/2	20/98	798/4256
	SVM_poly	1/2	19/98	790/4256
	SVM_rbf	6/10	18/90	1002/4256
	Softmax	0/0	22/100	907/4256
	ENV	4/7	17/93	922/4256
	BML	0/7	21/93	847/4256
7	None	-	2/100	24/1108
	SVM_linear	0/6	2/94	37/1108
	SVM_poly	0/6	2/94	37/1108
	SVM_rbf	6/10	0/90	57/1108
	Softmax	1/10	2/90	43/1108
	ENV	6/10	0/90	57/1108
	BML	6/10	0/90	57/1108
8	None	-	0/100	15/1120
	SVM_linear	0/10	0/90	14/1120
	SVM_poly	0/10	0/90	14/1120
	SVM_rbf	0/9	0/91	13/1120
	Softmax	0/6	0/94	9/1120
	ENV	0/9	0/91	16/1120
	BML	0/4	0/96	13/1120
9	None	-	7/100	118/1687
	SVM_linear	0/8	8/92	127/1687
	SVM_poly	0/9	8/91	129/1687
	SVM_rbf	1/10	6/90	96/1687
	Softmax	0/0	7/100	118/1687
	ENV	0/9	6/91	88/1687
	BML	4/9	3/91	153/1687
Benign	None	-	2/900	67/18491
	SVM_linear	1/41	1/859	72/18491
	SVM_poly	1/41	1/859	72/18491
	SVM_rbf	0/12	2/888	70/18491
	Softmax	2/49	0/851	108/18491
	ENV	2/22	0/878	95/18491
	BML	3/40	0/860	105/18491

Table 7 and 8 shows the classification result of the trained model. In column "Train False Rate" indicates the false rate of training data. In column "Test False Rate" indicates

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

the false rate of testing data.

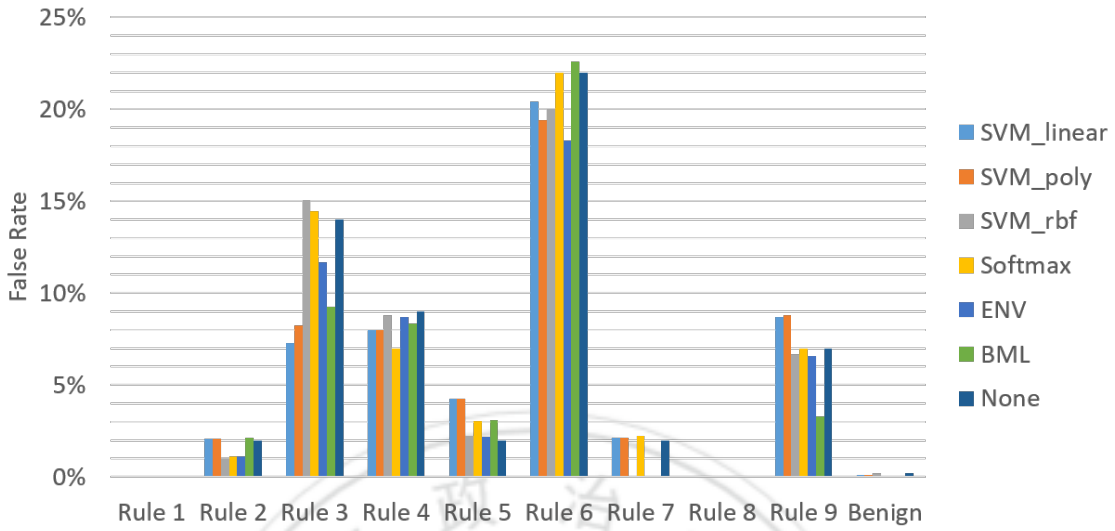


Figure 12: False rate of different majority training softmax neuron network on training data. (100*2 samples)

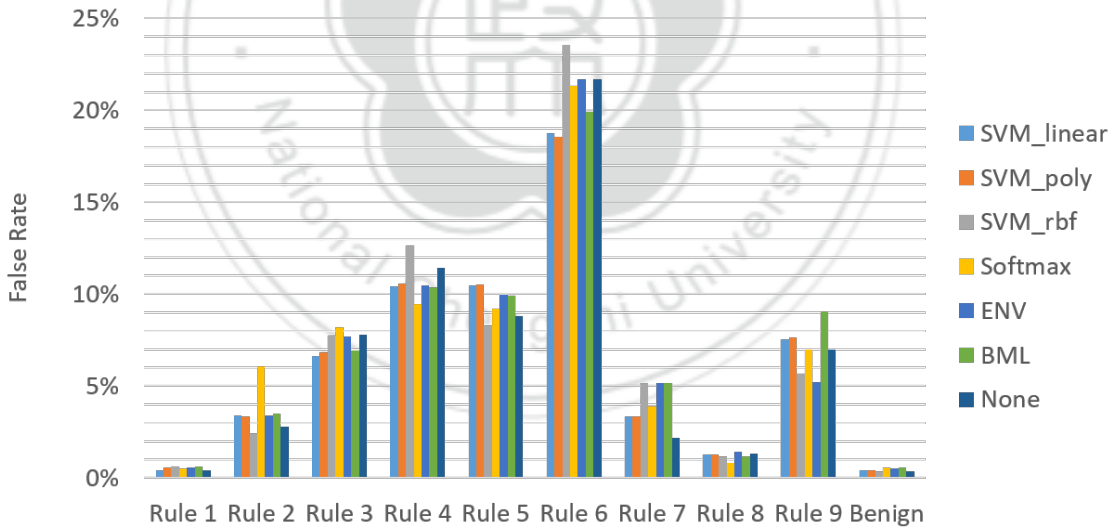


Figure 13: False rate of different majority training softmax neuron network on testing data. (100*2 samples)

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

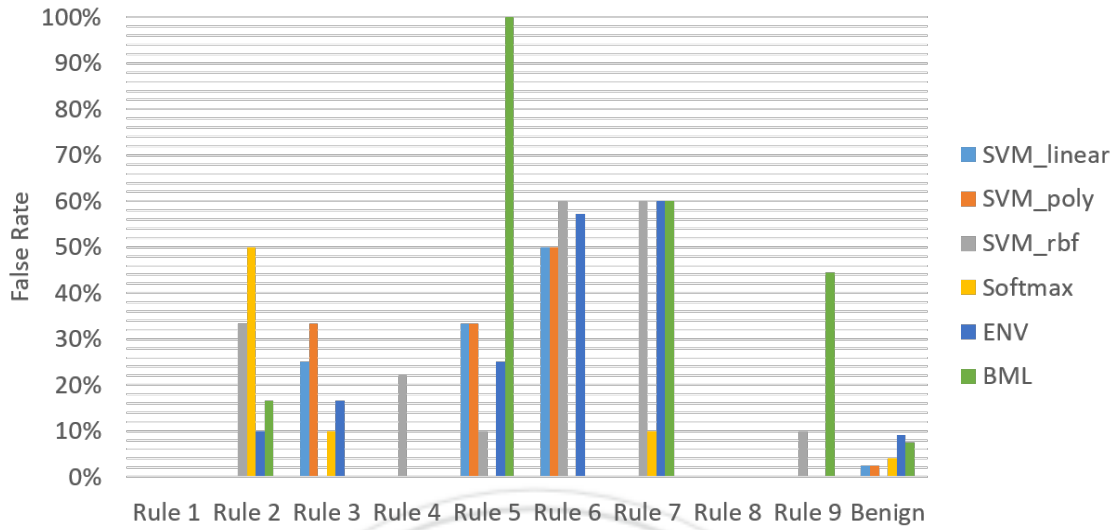


Figure 14: False rate of different majority training softmax neuron network on outlier data. (100*2 samples)

Figure 12 to Figure 14 show the false rate of softmax neural networks trained by different majorities. On average, all of the majority learning methods increase the classification accuracy of training data and do not loss much accuracy on testing data.

To answer RQ2, BML can choose proper majority set to make training data classified by softmax neural network more accurately. But, the 95% of majority set loss some information so that the accuracy of the classification on testing data is slightly lower than using all training data to train the softmax neural network.

4.2.3 Exp. 2.1: Majority Learning on Large Scale Data

RQ 3: How BML performs in terms of efficiency and accuracy compared to the state-of-the-art approaches on large-size data sets?

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

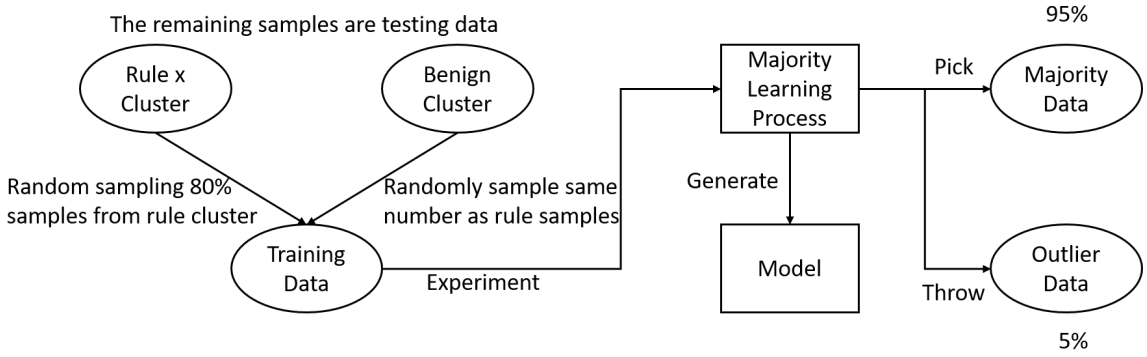


Figure 15: Experimental design of experiment 2.1.

Figure 15 illustrates the experimental design of experiment 2.1. In this subsection, we want to test the performance of BML learning a larger amount of data. We use 80% of the rule samples, rather than using randomly selected 100*2 (i.e., 100 benign + 100 malicious) samples.

For example, we would randomly select 80% of malware samples from the rule cluster 1 (3175 * 0.8 = 2540) and randomly select 2,540 benign samples from the benign cluster. Then, we labeled these 5,080 samples according to their class and regarded these samples as training set. The rest of the rule 1 cluster samples and the rest of the benign cluster samples would be the testing set. After we sampling the data from each clusters, we conduct the different majority learning experiments with the same sampling data.

Note that we also use the same amount of benign and malicious samples for training. The majority rate is set to 95%.

Table 9 shows the training result of the large-scale training set. In this set, BML reduces significantly training time compared to ENV. Also, BML does not need to increase model complexity for learning the 95% of training data. BML can train a proper SLFN for classifying the majority of bipartite data more efficiency than ENV. Figure 16 to Figure 18 show the false rate of SVM, softmax, ENV and BML. We calculate the mean false rate of 9 rules as same as exp 1-1. The accuracy performance of BML is between softmax and ENV. Figure 19 and shows the execution time of SVM, softmax, ENV and BML. ENV has slightly higher classification accuracy when the training data amount is larger, but the trade-off is the long model training time.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

To answer RQ3, BML is more time efficiency and does not lose much classification accuracy compared to the state-of-the-art approaches on large-size data sets.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

Table 9: Training result by using 80% samples

Rule	Majority Method	#HN	Outliers (#FB/#B, #FM/#M)	Execute Time(s)	Train B #FB/#B	Train M #FM/#M	Test B #FB/#B	Test M #FM/#M
1	SVM_linear	-	(60/186, 0/68)	70	0/2354	0/2472	490/16851	0/635
	SVM_poly	-	(60/185, 0/69)	66	0/2355	0/2471	490/16851	0/635
	SVM_rbf	-	(251/254, 0/0)	240	0/2286	0/2540	1773/16851	0/635
	Softmax	-	(39/113, 2/141)	95	0/2427	0/2399	319/16851	0/635
	ENV	1163	(83/157, 7/97)	9260	0/2383	0/2443	576/16851	4/635
	BML	1	(106/214, 7/40)	69	0/2326	0/2500	673/16851	2/635
2	SVM_linear	-	(0/47, 0/60)	6	0/1017	0/1004	12/18327	0/267
	SVM_poly	-	(0/46, 0/61)	6	0/1018	0/1003	18/18327	0/267
	SVM_rbf	-	(0/0, 55/107)	31	0/1064	0/957	0/18327	16/267
	Softmax	-	(0/2, 13/105)	11	0/1062	0/959	14/18327	3/267
	ENV	401	(0/16, 4/91)	2477	0/1048	0/973	6/18327	0/267
	BML	1	(0/54, 4/53)	11	0/1010	0/1011	0/18327	2/267
3	SVM_linear	-	(3/161, 0/1)	26	0/1459	0/1619	13/17771	0/405
	SVM_poly	-	(3/156, 0/6)	23	0/1464	0/1614	16/17771	0/405
	SVM_rbf	-	(0/0, 140/162)	76	0/1620	0/1458	0/17771	34/405
	Softmax	-	(2/12, 0/150)	34	0/1608	0/1470	2/17771	0/405
	ENV	175	(0/82, 0/80)	1461	0/1538	0/1540	2/17771	0/405
	BML	1	(0/131, 0/31)	23	0/1489	0/1589	2/17771	0/405
4	SVM_linear	-	(1/16, 0/131)	22	0/1454	0/1339	9/17921	0/368
	SVM_poly	-	(1/15, 0/132)	21	0/1455	0/1338	15/17921	0/368
	SVM_rbf	-	(0/0, 30/147)	72	0/1470	0/1323	0/17921	5/368
	Softmax	-	(3/15, 0/132)	28	0/1455	0/1338	31/17921	0/368
	ENV	667	(0/30, 0/117)	6762	0/1440	0/1353	1/17921	0/368
	BML	1	(0/69, 0/78)	20	0/1401	0/1392	1/17921	0/368
5	SVM_linear	-	(3/183, 0/13)	36	0/1777	0/1947	12/17431	0/491
	SVM_poly	-	(4/183, 0/13)	33	0/1777	0/1947	15/17431	0/491
	SVM_rbf	-	(0/141, 15/55)	117	0/1819	0/1905	0/17431	7/491
	Softmax	-	(30/132, 0/64)	46	0/1828	0/1896	330/17431	0/491
	ENV	1027	(0/41, 0/155)	8822	0/1919	0/1805	0/17431	0/491
	BML	1	(0/127, 0/69)	35	0/1833	0/1891	0/17431	0/491
6	SVM_linear	-	(2/136, 0/213)	116	0/3348	0/3271	8/15907	0/872
	SVM_poly	-	(3/125, 0/224)	128	0/3359	0/3260	14/15907	0/872
	SVM_rbf	-	(0/127, 51/222)	365	0/3357	0/3262	0/15907	18/872
	Softmax	-	(0/1, 0/348)	131	0/3483	0/3136	0/15907	0/872
	ENV	1399	(0/60, 0/289)	10724	0/3424	0/3195	0/15907	0/872
	BML	1	(0/166, 0/183)	136	0/3318	0/3301	0/15907	0/872
7	SVM_linear	-	(0/27, 0/70)	5	0/939	0/896	21/18425	0/242
	SVM_poly	-	(0/28, 0/69)	5	0/938	0/897	34/18425	0/242
	SVM_rbf	-	(0/0, 92/97)	32	0/966	0/869	0/18425	17/242
	Softmax	-	(7/97, 0/0)	8	0/869	0/966	298/18425	0/242
	ENV	121	(0/7, 0/90)	527	0/959	0/876	17/18425	0/242
	BML	1	(0/23, 0/74)	10	0/943	0/892	14/18425	0/242
8	SVM_linear	-	(3/19, 0/79)	5	0/957	0/897	13/18415	0/244
	SVM_poly	-	(3/15, 0/83)	4	0/961	0/893	16/18415	0/244
	SVM_rbf	-	(0/0, 25/98)	24	0/976	0/878	0/18415	12/244
	Softmax	-	(18/58, 0/40)	8	0/918	0/936	220/18415	0/244
	ENV	511	(0/33, 0/65)	6248	0/943	0/911	2/18415	0/244
	BML	1	(0/76, 0/22)	9	0/900	0/954	7/18415	0/244
9	SVM_linear	-	(1/132, 0/11)	18	0/1297	0/1418	15/17962	0/358
	SVM_poly	-	(1/133, 0/10)	19	0/1296	0/1419	20/17962	0/358
	SVM_rbf	-	(0/0, 62/143)	59	0/1429	0/1286	0/17962	17/358
	Softmax	-	(73/97, 0/46)	22	0/1332	0/1383	953/17962	0/358
	ENV	497	(0/29, 0/114)	5639	0/1400	0/1315	0/17962	0/358
	BML	1	(0/82, 0/61)	17	0/1347	0/1368	0/17962	0/358

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

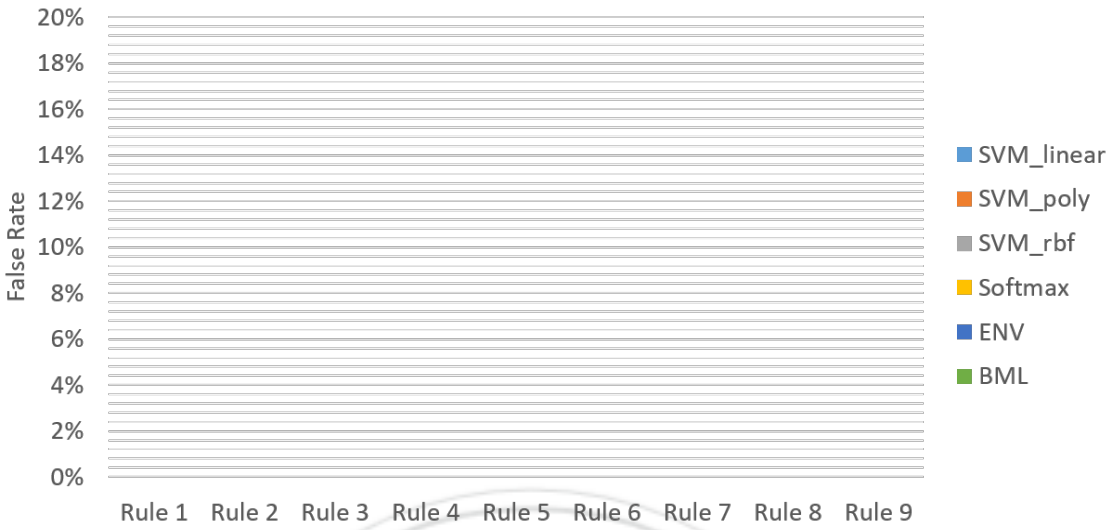


Figure 16: False rate of different majority learning methods on training data (80% samples).

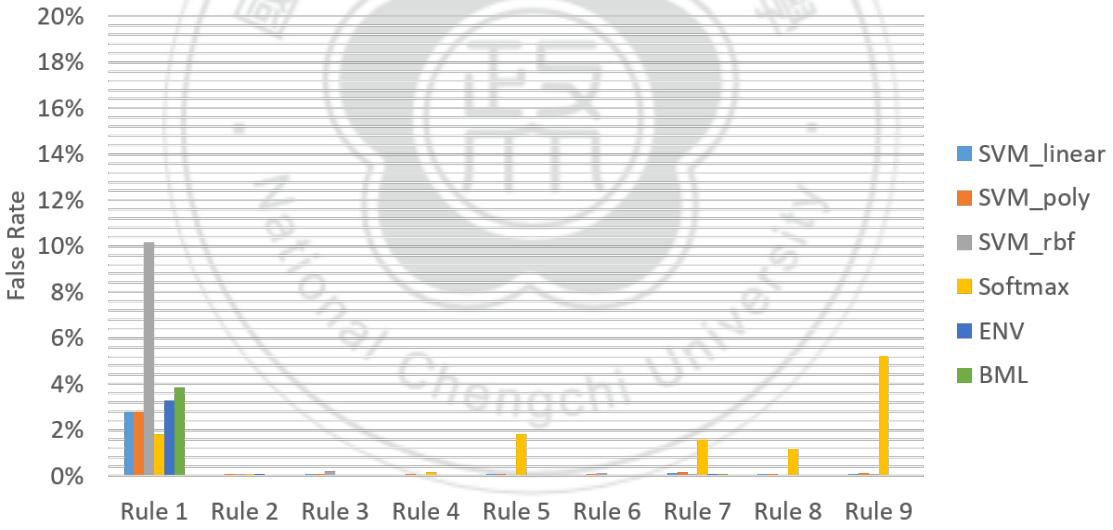


Figure 17: False rate of different majority learning methods on testing data (80% samples).

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

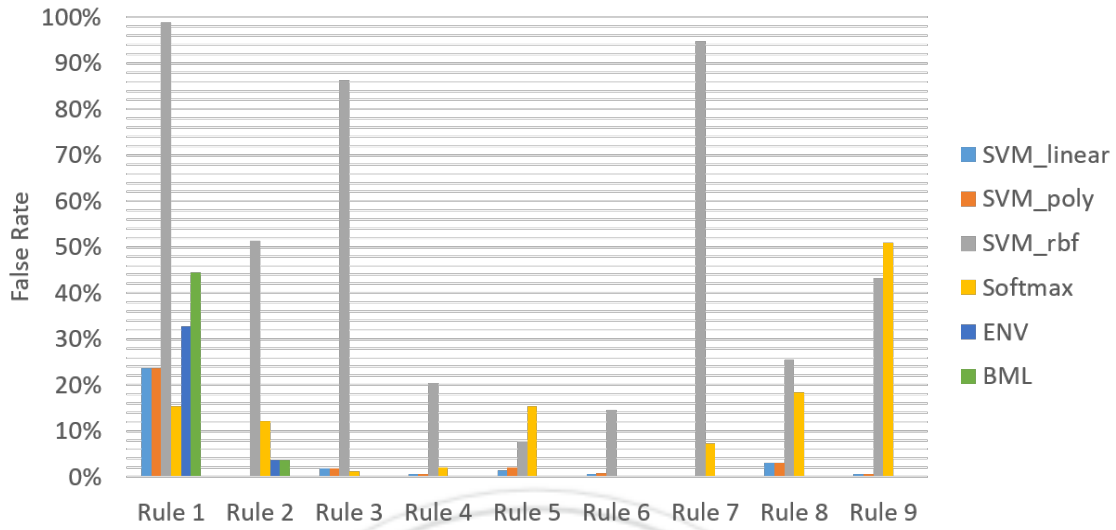


Figure 18: False rate of different majority learning methods on outlier data (80% samples).

Figure 19: Execution time of different majority learning methods (80% samples).

4.2.4 Exp. 2.2: Use ANN to Learn the Larger Amount of Majority

RQ 4: How well the majority set selected by BML represents on large-size data sets?

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

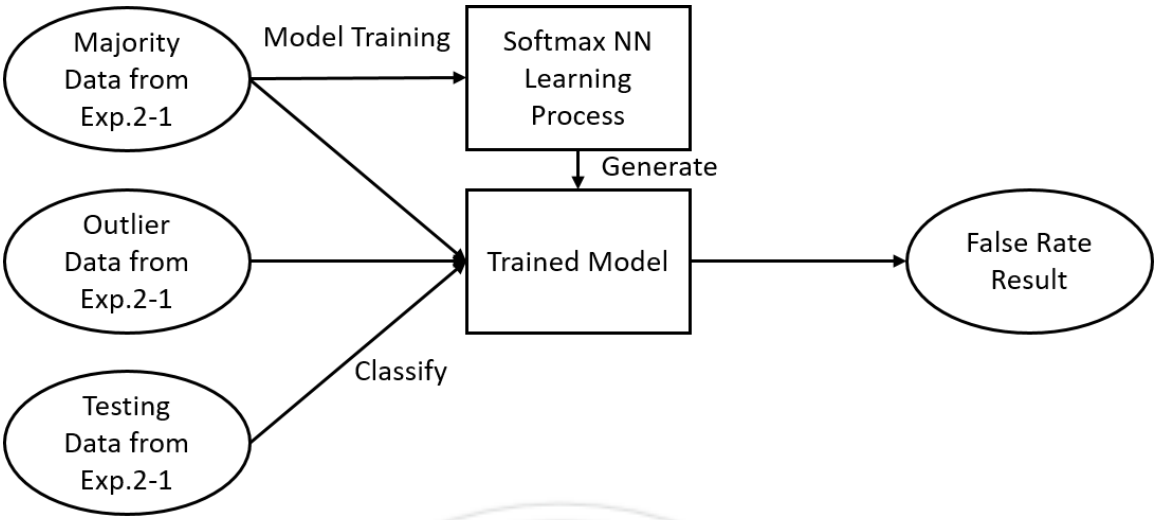


Figure 20: Experimental design of experiment 2.2.

Figure 20 illustrates the experimental design of experiment 2.2. We took the same approach as experiment 1.2, but this time we used the majority of 80% of rule data. We also test softmax learning by directly using the original 80% of rule data. The softmax neural network classification result on large-scale data is listed in Table 10 and 11.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

Table 10: Softmax neural network classification result by using 80% samples

Rule	Majority Method	Outlier FR	Train FR	Test FR
1	None	-	5/2540	1/635
	SVM_linear	1/68	5/2472	2/635
	SVM_poly	2/69	4/2471	2/635
	SVM_rbf	0/0	4/2540	1/635
	Softmax	0/141	4/2399	1/635
	ENV	1/97	3/2443	1/635
	BML	1/40	3/2500	1/635
2	None	-	195/1064	39/267
	SVM_linear	47/60	202/1004	50/267
	SVM_poly	48/61	202/1003	50/267
	SVM_rbf	45/107	189/957	49/267
	Softmax	21/105	182/959	44/267
	ENV	35/91	175/973	44/267
	BML	30/53	192/1011	46/267
3	None	-	25/1620	3/405
	SVM_linear	0/1	20/1619	3/405
	SVM_poly	0/6	20/1614	3/405
	SVM_rbf	2/162	22/1458	3/405
	Softmax	37/150	9/1470	10/405
	ENV	3/80	22/1540	3/405
	BML	1/31	24/1589	4/405
4	None	-	252/1470	64/368
	SVM_linear	35/131	245/1339	69/368
	SVM_poly	35/132	245/1338	70/368
	SVM_rbf	22/147	260/1323	69/368
	Softmax	89/132	204/1338	74/368
	ENV	48/117	231/1353	71/368
	BML	0/78	233/1392	60/368
5	None	-	351/1960	89/491
	SVM_linear	13/13	344/1947	85/491
	SVM_poly	12/13	345/1947	82/491
	SVM_rbf	8/55	317/1905	75/491
	Softmax	4/64	357/1896	88/491
	ENV	54/155	277/1805	77/491
	BML	69/69	319/1891	93/491

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

Table 11: Softmax neural network classification result by using 80% samples (cont.)

Rule	Majority Method	Outlier FR	Train FR	Test FR
6	None	-	78/3484	25/872
	SVM_linear	4/213	86/3271	28/872
	SVM_poly	8/224	89/3260	30/872
	SVM_rbf	28/222	56/3262	31/872
	Softmax	0/348	47/3136	19/872
	ENV	30/289	53/3195	30/872
	BML	0/183	71/3301	23/872
7	None	-	42/966	6/242
	SVM_linear	37/70	18/896	9/242
	SVM_poly	37/69	18/897	9/242
	SVM_rbf	51/97	5/869	9/242
	Softmax	0/0	42/966	7/242
	ENV	39/90	14/876	8/242
	BML	39/74	14/892	9/242
8	None	-	46/976	13/244
	SVM_linear	0/79	46/897	13/244
	SVM_poly	0/83	46/893	13/244
	SVM_rbf	18/98	34/878	13/244
	Softmax	1/40	44/936	13/244
	ENV	10/65	36/911	13/244
	BML	0/22	41/954	13/244
9	None	-	804/1429	204/358
	SVM_linear	0/11	681/1418	175/358
	SVM_poly	0/10	669/1419	172/358
	SVM_rbf	65/143	754/1286	212/358
	Softmax	3/46	699/1383	184/358
	ENV	44/114	749/1315	206/358
	BML	42/61	729/1368	203/358
Benign	None	-	63/15509	11/3882
	SVM_linear	77/907	32/14602	24/3882
	SVM_poly	79/886	32/14623	25/3882
	SVM_rbf	36/522	47/14987	11/3882
	Softmax	52/527	37/14982	12/3882
	ENV	24/455	48/15054	13/3882
	BML	31/942	48/14567	13/3882

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

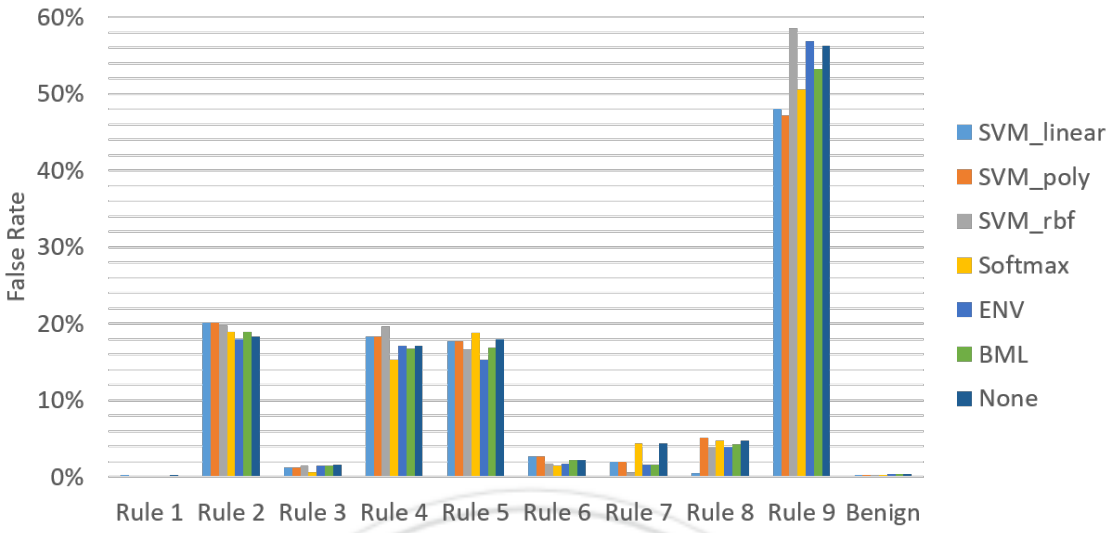


Figure 21: False rate of different majority training softmax neuron network on training data. (80% samples)

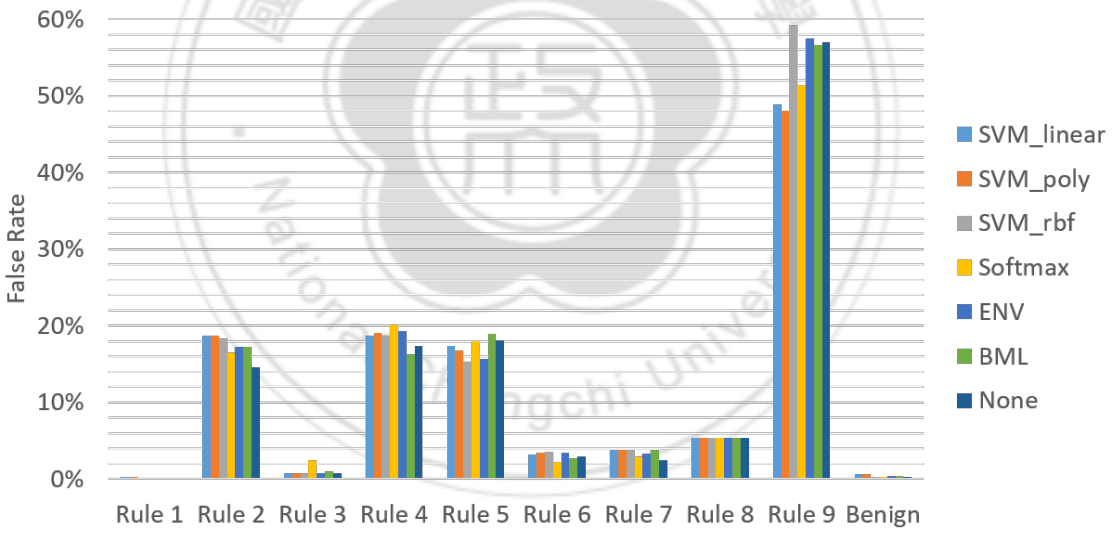


Figure 22: False rate of different majority training softmax neuron network on testing data. (80% samples)

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

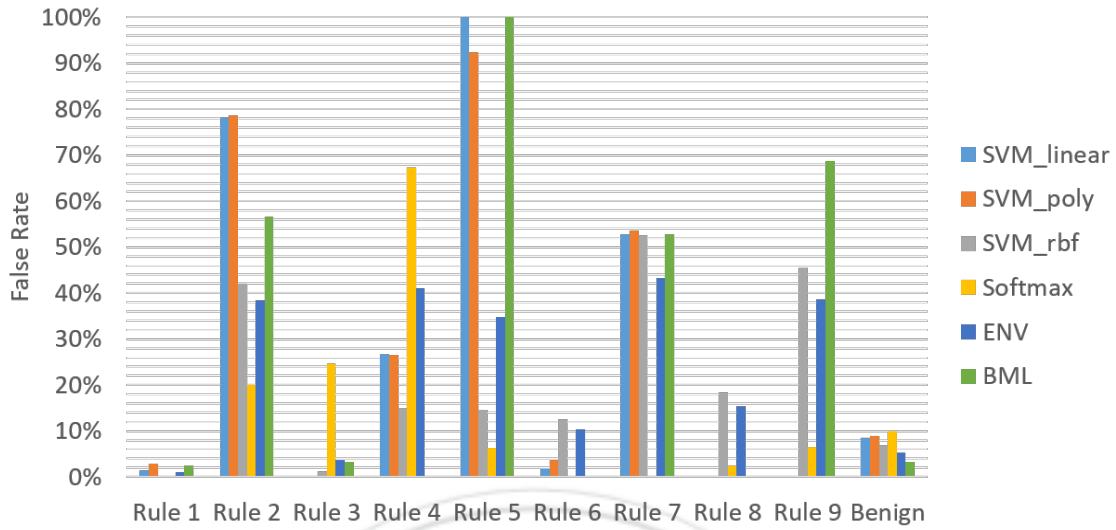


Figure 23: False rate of different majority training softmax neuron network on outlier data. (80% samples)

Figure 21 to Figure 23 show the false rate of softmax, ENV and BML. The majority data selected by the different majority learning methods increase the learning effect of softmax neural network. However, compared to experiment 1-2, the classification accuracy of using small size data to train the model is much higher than using large size data. We applied gradient descent 10,000 times for the model in experiment 1-2 and 2-2. We speculate that it may be because when the amount of data is large, the softmax neural network needs to do more weight tuning to reach the same accuracy.

To answer RQ4, BML can choose proper majority set to make training data classified by softmax neural network more accurately.

In experiment 1 and experiment 2, we performed the majority learning for each cluster. Benign samples are more possible to be outliers. After GHSOM clustering, the similarities of malicious samples are relatively higher than benign ones; therefore, we found that malicious samples would be less likely to be misclassified as benign ones. We speculate that the pattern of malicious behavior is more consistent, so BML is easier to learn the taxonomy.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

4.2.5 Exp. 3: Binary Classification Performance

RQ5: How BML performs on high-variation data sets?

The above experiments have proved the effectiveness of BML classifying rule samples and benign samples. However, the rule samples are clustered by GHSOM, which means the features of rule samples in the same cluster are similar.

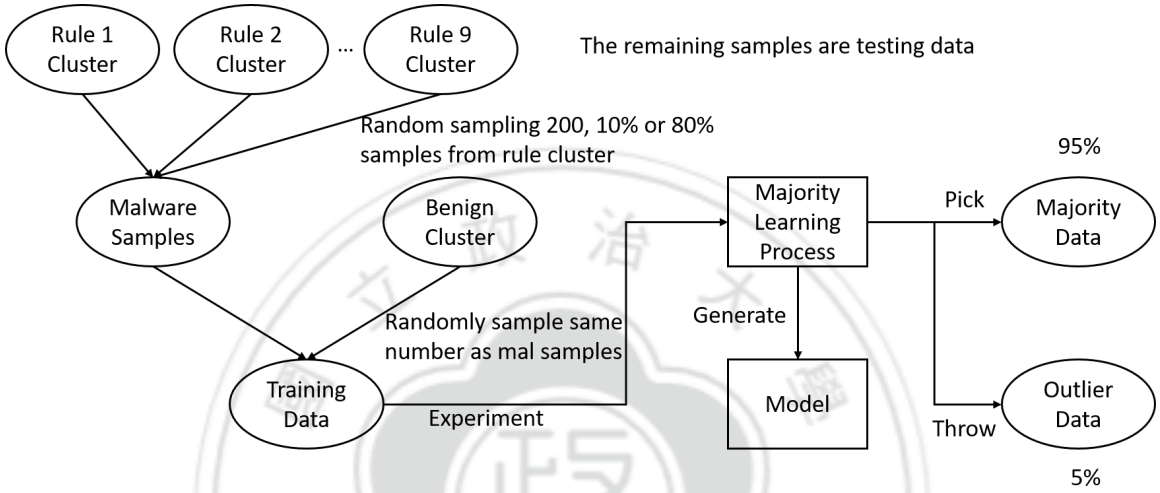


Figure 24: Experimental design of experiment 3.1.

Figure 24 illustrates the experimental design of experiment 3.1. In this experiment, we want to test whether BML can learn good binary classification criteria under the condition that the malicious behavior patterns are more complicated. We consider all kinds of malicious clusters as one malicious category and conduct our bipartite learning on the two kinds but spread samples, i.e., benign and malicious. We use three methods to build up the training data: 200, 10% and 80%. "200" means that we randomly select 200 malware samples from 9 rules separately, and then randomly select an equal amount of benign samples (i.e. 1,800 benign samples). "10%" means that we randomly select 10% malware samples from 9 rules separately, and then randomly select an equal amount of benign samples. "80%" means that we randomly select 80% malware samples from 9 rules separately, and then randomly select an equal amount of benign samples. Note that the same amount of benign samples are randomly selected and used for training as well.

We used ENV and softmax neural network as the experimental control group. The

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

terminal condition of ENV and softmax neural network is the neural network can correctly classify 95% of the training data. We compare the performance of BML with the other two majority learning methods.

Table 12: Training result of bipartite classification.

Sampling Method	Majority Method	#HN	Outliers (#FB/#B, #FM/#M)	Execute Time(s)	Train B #FB/#B	Train M #FM/#M	Test B #FB/#B	Test M #FM/#M
200	SVM_linear	-	(7/7, 161/173)	31	0/1793	0/1627	57/17591	889/17591
	SVM_poly	-	(7/8, 153/172)	33	0/1792	0/1628	53/17591	792/17591
	SVM_rbf	-	(152/180 0/0)	133	0/1620	0/1800	1359/17591	0/17591
	Softmax	-	(33/77, 0/103)	37	0/1723	0/1697	278/17591	1/17591
	ENV	3429	(5/5, 170/175)	9010	0/1795	0/1625	1295/17591	5526/17591
	BML	107	(54/162, 12/18)	1205	0/1638	0/1782	589/17591	95/17591
10%	SVM_linear	-	(3/3, 169/191)	32	0/1932	0/1744	53/17456	1469/17456
	SVM_poly	-	(3/3, 189/191)	38	0/1932	0/1744	57/17456	1763/17456
	SVM_rbf	-	(1/1 72/193)	690	0/1934	0/1742	47/17456	757/17456
	Softmax	-	(7/12, 137/182)	230	0/1923	0/1753	87/17456	1214/17456
	ENV	401	(51/51, 139/143)	6100	0/1884	0/1792	38/17456	5869/17456
	BML	1	(32/35, 159/159)	33	0/1900	0/1776	347/17456	1446/17456
80%	SVM_linear	-	(555/590, 771/961)	3108	0/14919	0/14548	194/3882	889/3882
	SVM_poly	-	(573/846, 11/705)	3139	0/14663	0/14804	150/3882	4/3882
	SVM_rbf	-	(1439/1551 0/0)	5247	0/13958	0/15509	361/3882	0/3882
	Softmax	-	(416/750, 12/801)	2031	0/14759	0/14708	100/3882	4/3882
	ENV	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	BML	1	(551/1321, 0/230)	2316	0/14188	0/15279	133/3882	0/3882

Table 12 shows the training result of three sampling methods. The execution time of 80% ENV is longer than 20,000 seconds, so we didn't finish this experiment. Since the malicious patterns are more complex, ENV needs a lot of time for training. BML has higher time efficiency and lower model complexity compare to ENV. BML has much higher time efficiency than SVMs, but softmax has much higher time efficiency than BML.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

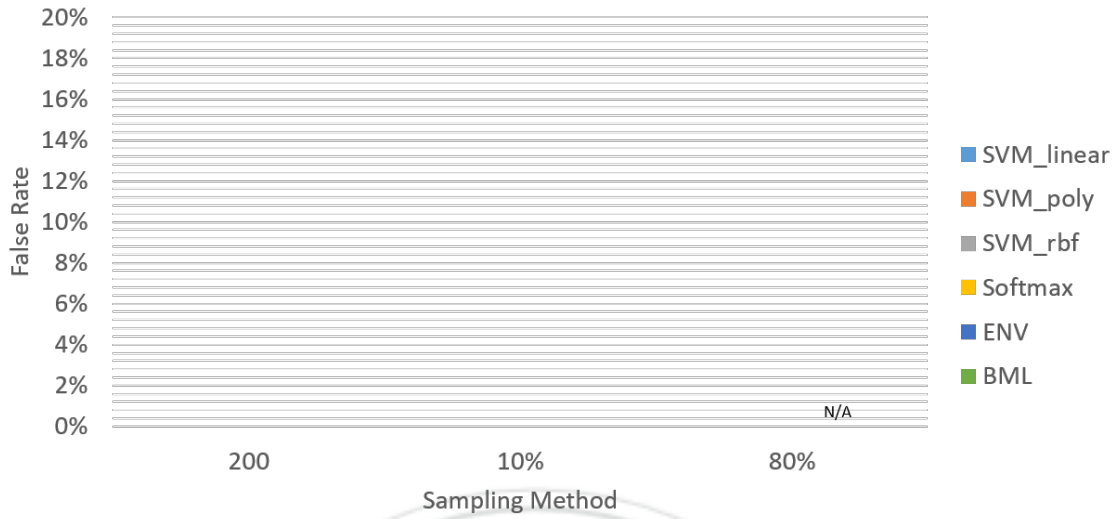


Figure 25: False rate of different majority learning methods on training data (variety samples).

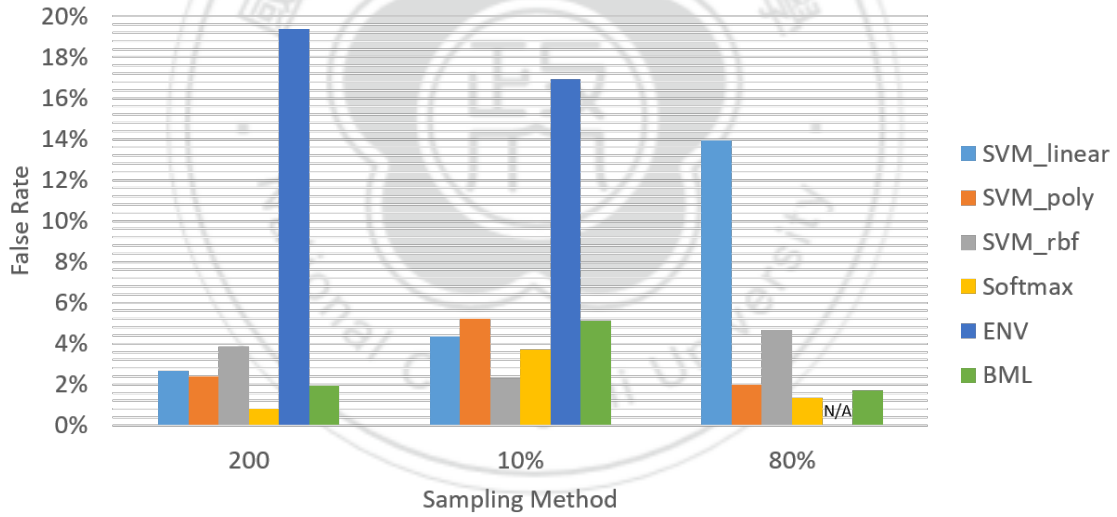


Figure 26: False rate of different majority learning methods on testing data (variety samples).

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

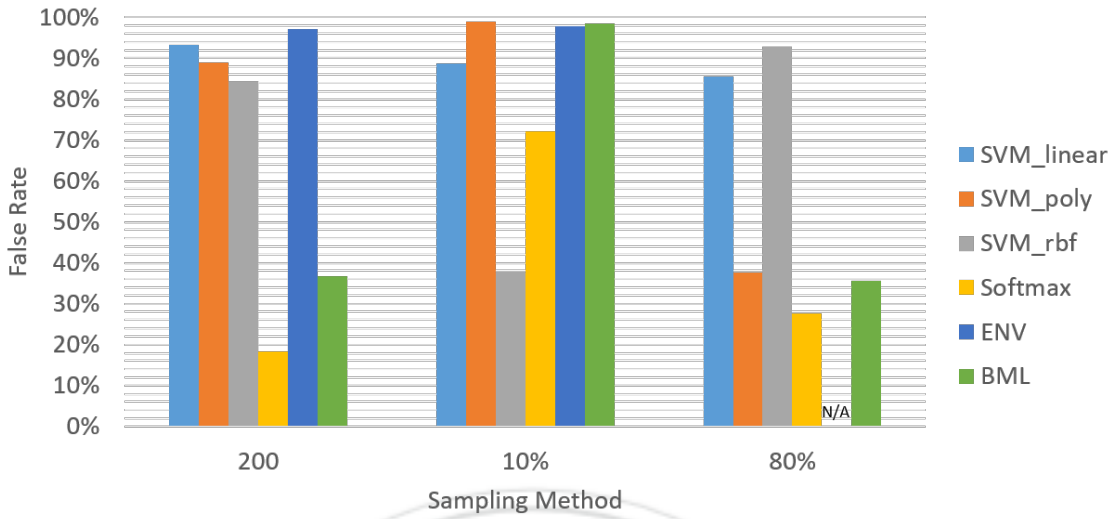


Figure 27: False rate of different majority learning methods on outlier data (variety samples).

Figure 25 to Figure 27 show the mean false rate of the three different sampling methods.

To answer RQ5, in the scenario of high variety data, BML can perform higher time efficiency and higher classification accuracy compared to ENV. But, softmax has more efficiency than BML.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

5 Discussion

In chapter 4, we have shown the experiment results. Generally speaking, BML performs high time efficiency and remain the same level classification accuracy compared with the state-of-art methods. In this chapter, we will explain the experiment results in more detail.

5.1 Exp. 1.1: Majority Learning on Small-Size Sampling Data

In Experiment 1.1, we randomly obtained 100 rule samples and 100 benign samples as training data. Compared to the number of all rule samples, 100 samples are less than 10% of the total number of samples. This small portion of data might not able to represent the distribution of the population. But, the rule sample have clustered by GHSOM algorithm, the rule sample should have similar pattern. That is to say, the 100 sampling rule samples are similar. Therefore, we should not worry too much about the representativeness of the obtained rule samples. As for the benign samples, there are 133 benign clusters clustered by GHSOM.

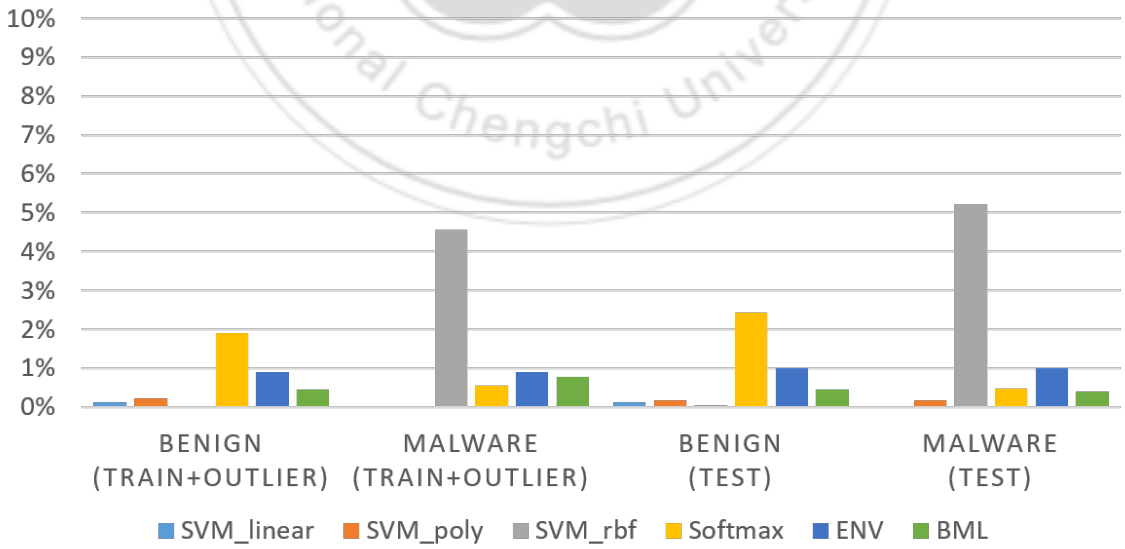


Figure 28: False rate of different majority learning methods (100*2 samples).

The variety of benign sample patterns are larger than the malware sample patterns. Therefore, we sampled as much randomly as possible so that the obtained 100 benign

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

samples are not overly concentrated on a few specific clusters. In figure 28, the false rate of training data and testing data are related. To a certain extent, the sampling 100 rule samples and 100 benign samples can represent the characteristics of the overall data.

We can discover that the execution time of ENV is much longer than softmax and BML in table 6. The main reason for the result is that ENV needs to retrain the model much more times than softmax and BML. ENV is a relatively strict majority learning method. Thus, ENV needs more model training procedure to fit the strict limitation. The model training procedure of ENV including weight-tuning and add hidden nodes. Since adding hidden nodes only needs to calculate newly hidden node weights, the time complexity is O(1), which means adding hidden nodes does not need too much time. The most time-consuming process is to use the gradient descent method tuning the weight. The gradient descent method needs to do forward and backward pass repeatedly so many times, this process needs a lot of computation power. When the hidden node amount increased, the model needs even more computation power to deal with the extra calculation. Table 6 shows that ENV indeed retrained the model and add hidden nodes in these 9 experiments. Therefore, ENV spent the most time on model training compared to softmax and BML.

The softmax majority learning method and BML train the model much faster because they do not need to retrain the model after the initialize of the model. Both methods obtained $m + 1$ samples for model initialization. The softmax majority learning method applies gradient descent method a few times to learn the $m + 1$ samples. BML adopts simultaneous equations to calculate proper initial weights for the $m + 1$ samples. After the $m + 1$ samples were learned by the model, the majority can be correctly classified by softmax and BML. Thus, softmax and BML save much time in retraining the model. We believe that the reason why BML and softmax do not need retraining is that the rule sample and the benign sample are the data clustered by the GHSOM algorithm, and the sample features have a certain degree of difference, so the malware and the benign samples are not difficult to be distinguished by the neural network.

Figure 28 shows the average false rate of the 9 experiments. The classification result

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

indicates that BML has the highest classification accuracy. However, BML is a less strict majority learning method compared to ENV, the classification accuracy of BML should not better than ENV. We hypothesise that the training data amount is not large enough, the number of samples taken is not sufficient to represent all the features of the population, so the BML classification results could be better than ENV.

5.2 Exp. 1.2: Use ANN to Learn the Majority

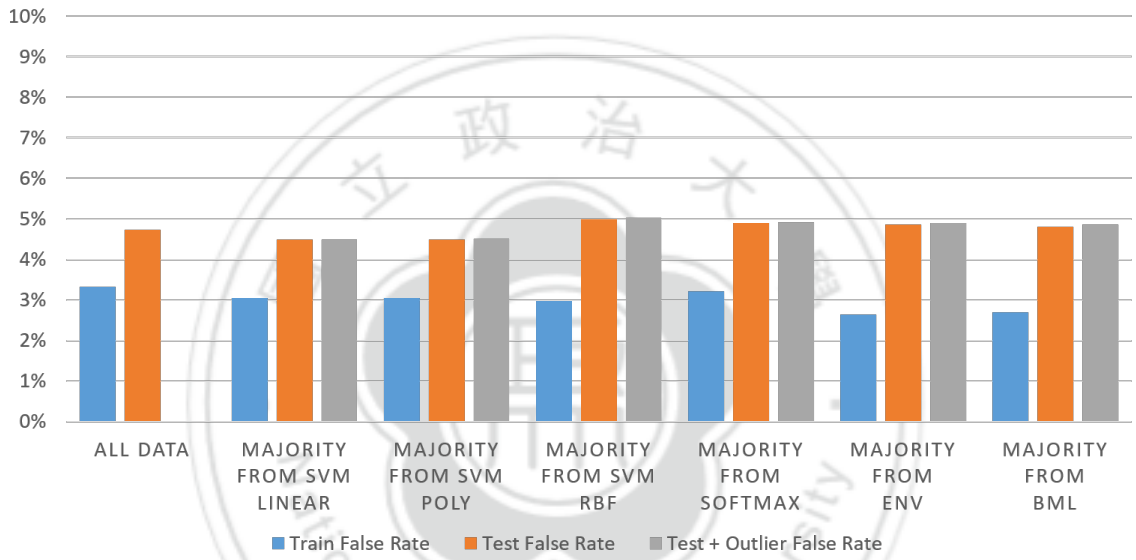


Figure 29: False rate of different softmax neuron network. (100*2 samples)

In experiment 1.2, we have tested the majority data selected by different majority learning methods. Figure 29 shows that all the majority learning methods can find the proper majority. Compared to using all data for training, the different selected majority data do not lose accuracy on testing data. In other words, the majority learning methods can help us to pick fewer data for model training but remain the same level classification accuracy. As for the training data, both ENV and BML can select proper majority to slightly increase classification accuracy. We can prove that the majority is selected properly by BML in the small size data set.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

5.3 Exp. 2.1: Majority Learning on Large Scale Data

Table 9 shows the training result of the different majority learning methods. Obviously, ENV needs lots of time for model training, and the hidden node amount is very large. The SLFNs grew too big, and thus extended the model training time. We have observed that once ENV needs to apply adding hidden nodes process, the later stages are almost inevitable to add hidden nodes. Pure back-propagation process cannot get rid of local optimal problem. Because of the newly hidden node weights were precisely calculated, the gradient descent method was unable to make appropriate adjustments for the neuron weights.

As for softmax and BML, the situation is the same as experiment 1.1: after the model was initialized, the majority can be correctly classified by softmax and BML. Both softmax and BML need to check the condition L for $(\gamma N - m - 1)$ times. The size of the training set in experiment 2.1 is larger, so the execution time in experiment 2.1 is more than experiment 1.1.

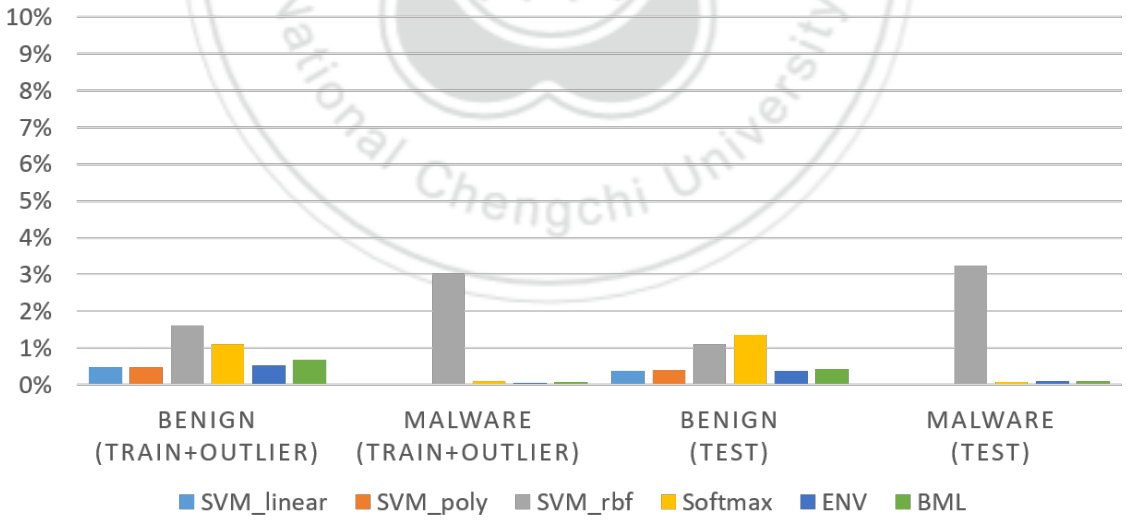


Figure 30: False rate of different majority learning methods (80% samples).

Figure 30 shows the false rate of the different majority learning methods. Compared to experiment 1.1, the classification accuracy became much higher. We surmise that the more data were used for training, the more information would be learned by the models. In addition, when the size of the training set is large enough, ENV has better performance on

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

classification accuracy compared to BML. However, the trade of is the long model training time. BML can learn the majority very fast while remaining competitive classification accuracy.

5.4 Exp. 2.2: Use ANN to Learn the Larger Amount of Majority

Figure 31: False rate of different softmax neuron network. (80% samples).

Figure 31 shows the false rate of ANN. The classification accuracy is lower than experiment 1.2. There are two possible reasons for this result. First, we adopt simple ANN in this experiment. That is to say, the ANN does not have any hidden layer. The simple model might not be able to learn the feature of training data such precisely when the size of training set is large. Another possible reason is that we apply gradient descent the same 10,000 times . When the data size becomes larger, ANN model might need more times weight tuning to achieve the same level of classification accuracy.

The majority learning methods indeed select the proper majority. Compared to using all data for training, the different majority learning methods have same level classification accuracy on testing data. Also, the outliers are more easily to be mis-classified by the ANN. As for the training accuracy, all of the majority learning methods can increase the

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

classification accuracy on training data compared to using all data. We can prove that BML can find proper majority on larger training set.

5.5 Exp. 3: Binary Classification Performance

In experiment 3, we designed 3 different sampling method. Sampling method "10%" and "80%" are the experiments for testing small and large size training set. Sampling method "200" is regarded as the experimental comparison of "10%", due to the similar in training set size.

Table 12 shows the training result. ENV has no doubt longest model training time. ENV grows many hidden nodes to fit the majority of the training data. Thus, ENV consumed lots of time for training a perfect model. However, figure 25 and figure 26 indicate that ENV has the worst classification accuracy. We supposed that the ENV models over-fit the training data. So, the trained models have poor classification accuracy on malware testing data.

We mentioned in section 3.3 that the classification of softmax is actually a variant of the condition L. So, softmax and BML are no doubt having similar classification accuracy. As for the model training time, we will discuss the three different sampling methods separately.

In the experiment sampling method "200", softmax only tuning the weights 3,600 times while BML cramming 53 times and tuning the weights 132,802 times. Thus, the execution time is separately 37 and 1,205 seconds, softmax has more time efficient than BML.

In the experiment sampling method "10%", softmax tuning the weights 34,971 times while BML does not need any training. BML only needs time for checking majority. Thus, the execution time is separately 230 and 33 seconds, BML has more time efficient than softmax.

In the experiment sampling method "80%", softmax tuning the weights 2,023 times while BML does not need any training. Although BML only needs time for checking majority, softmax has a relatively simple model. Forward pass calculation is easier for

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

softmax model. Thus, the execution time is separately 2,031 and 2,316 seconds, softmax has more time efficient than BML.

5.6 Majority Learning on SVMs

We have mentioned in section 3.4 that SVM have four prevalent kernel for data classification, Linear, polynomial, RBF and sigmoid. In general, we should choose the appropriate kernel based on the characteristics of the data distribution. We had adopted all the four kernel functions for majority learning. However, the sigmoid kernel has poor ability for classifying the system call data in our experiment. So, we did not list the result of SVM majority learning with sigmoid kernel.

In experiment 1.2 and 2.2, we have proof that the SVM majority learning with linear, polynomial and RBF kernel can choose proper majority just like other majority learning methods.

In experiment 1.1 and 2.1, the SVM majority learning methods with linear and polynomial kernel have similar performance on classification accuracy and time efficiency. These two methods have on average the best performance compare to other majority learning methods. We speculate that the reason for this experimental result is that the system call data set distribution is proper for using linear and polynomial kernel for classifying two class of data. As for the SVM majority learning methods with RBF kernel, it has on average the worst classification accuracy compare to other majority learning methods. We can draw a conclusion that choosing a proper kernel for classifying the data set is important.

In experiment 3.1, the classification accuracy of SVMs is on average worse than BML. We supposed that when the training data has more variety, SVMs are more difficult to find a hyperplane to separate two class of data. Relatively speaking, BML is a more stable majority learning method.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

6 Conclusion

Noisy labels are almost inevitable in real-world cases. In this paper, we introduce a novel nominal resistant learning procedure BML to avoid anomalies affecting the effectiveness of learning. Through picking the observations with the maximum distance of two classes, we are able to spot anomalies in a global view when the feature of anomalies is unknown. Further, we apply the resistant learning mechanism to reduce the impact of outliers on neural networks.

We had applied the majority learning concept on other prevalent classification models, SVMs and ANN. Through selecting the data which is familiar to the models, these majority learning methods can learn proper majority and avoid the interference of outliers.

Besides the optimization on the algorithm, We implemented the majority learning algorithms with TensorFlow and executed in GPU environment to accelerate the model training process.

Experiments on real-world data sets show that our approach has a classification accuracy similar to the envelope mechanism but have more time efficiency. We also use popular multi-class classification model, the softmax neural network, to learn the majority selected by BML and perform a higher classification accuracy on the training data, and remain the same level classification accuracy on the testing data.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

References

[1] W. Huang, Y. Yang, Z. Lin, G.-B. Huang, J. Zhou, Y. Duan, and W. Xiong, "Random feature subspace ensemble based extreme learning machine for liver tumor detection and segmentation," in *Engineering in Medicine and Biology Society (EMBC), 2014 36th Annual International Conference of the IEEE*, pp. 4675–4678, IEEE, 2014.

[2] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE transactions on neural networks*, vol. 8, no. 1, pp. 98–113, 1997.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[4] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1725–1732, 2014.

[5] G.-B. Huang and H. A. Babri, "Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions," *IEEE Transactions on Neural Networks*, vol. 9, no. 1, pp. 224–229, 1998.

[6] F. Anscombe, "Graphs in statistical analysis," *The American Statistician*, vol. 27, no. 1, pp. 17–21, 1973.

[7] R.-H. Tsaih and T.-C. Cheng, "A resistant learning procedure for coping with outliers," *Annals of Mathematics and Artificial Intelligence*, vol. 57, no. 2, pp. 161–180, 2009.

[8] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM computing surveys (CSUR)*, vol. 44, no. 2, p. 6, 2012.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

[9] S.-Y. Huang, F. Yu, R.-H. Tsaih, and Y. Huang, "Resistant learning on the envelope bulk for identifying anomalous patterns," in *Neural Networks (IJCNN), 2014 International Joint Conference on*, pp. 3303–3310, IEEE, 2014.

[10] "TensorFlow." `https://www.tensorflow.org/`.

[11] G.-B. Huang, Y.-Q. Chen, and H. A. Babri, "Classification ability of single hidden layer feedforward neural networks," *IEEE Transactions on Neural Networks*, vol. 11, no. 3, pp. 799–801, 2000.

[12] R. Tsaih, "The softening learning procedure," *Mathematical and computer modelling*, vol. 18, no. 8, pp. 61–64, 1993.

[13] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: a new learning scheme of feedforward neural networks," in *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, vol. 2, pp. 985–990, IEEE, 2004.

[14] G. Feng, G.-B. Huang, Q. Lin, and R. Gay, "Error minimized extreme learning machine with growth of hidden nodes and incremental learning," *IEEE Transactions on Neural Networks*, vol. 20, no. 8, pp. 1352–1357, 2009.

[15] P. J. Rousseuw and A. M. Leroy, "Robust regression and outlier detection," 1987.

[16] A. C. Atkinson, "Plots, transformations and regression; an introduction to graphical methods of diagnostic regression analysis," tech. rep., 1985.

[17] R. D. Cook and S. Weisberg, *Residuals and influence in regression*. New York: Chapman and Hall, 1982.

[18] J. Law, "Robust statistics-the approach based on influence functions.," 1986.

[19] Y. Ren, P. Zhao, Y. Sheng, D. Yao, and Z. Xu, "Robust softmax regression for multi-class classification with self-paced learning," in *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pp. 2641–2647, 2017.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

[20] W. Jiang, H. Gao, F.-l. Chung, and H. Huang, "The l2, 1-norm stacked robust autoencoders for domain adaptation.," in *AAAI*, pp. 1723–1729, 2016.

[21] C. Zhou and R. C. Paffenroth, "Anomaly detection with robust deep autoencoders," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 665–674, ACM, 2017.

[22] H. Zhao and Y. Fu, "Semantic single video segmentation with robust graph representation.," in *IJCAI*, pp. 2219–2226, 2015.

[23] D. Wang and X. Tan, "Robust distance metric learning in the presence of label noise.," in *AAAI*, pp. 1321–1327, 2014.

[24] Z. Jia and H. Zhao, "A joint graph model for pinyin-to-chinese conversion with typo correction," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, pp. 1512–1523, 2014.

[25] P. J. Huber, "Robust statistics. 1981."

[26] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[27] S. Hou, Y. Ye, Y. Song, and M. Abdulhayoglu, "Hindroid: An intelligent android malware detection system based on structured heterogeneous information network," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1507–1515, ACM, 2017.

[28] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial perturbations against deep neural networks for malware classification," *arXiv preprint arXiv:1606.04435*, 2016.

[29] Q. Wang, W. Guo, K. Zhang, A. G. Ororbia II, X. Xing, X. Liu, and C. L. Giles, "Adversary resistant deep neural networks with an application to malware detection,"

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1145–1153, 2017.

[30] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 3422–3426, IEEE, 2013.

[31] C.-H. Chiu, J.-J. Chen, and F. Yu, "An effective distributed ghsom algorithm for unsupervised clustering on big data," in *Big Data (BigData Congress), 2017 IEEE International Congress on*, pp. 297–304, IEEE, 2017.

[32] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[33] R. M. Bell and Y. Koren, "Lessons from the netflix prize challenge," *Acm Sigkdd Explorations Newsletter*, vol. 9, no. 2, pp. 75–79, 2007.

[34] S. Haykin, *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[35] "TensorFlow - MNIST For ML Beginners." `https://www.tensorflow.org/versions/r1.1/get_started/mnist/beginners`.

[36] A. R. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Transactions on Information theory*, vol. 39, no. 3, pp. 930–945, 1993.

[37] "TensorFlow - tf.matrix_solve_ls." `https://www.tensorflow.org/api_docs/python/tf/matrix_solve_ls`.

[38] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM transactions on intelligent systems and technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[39] "Scikit-Learn Support Vector Machines." `https://scikit-learn.org/stable/modules/svm.html`.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05

[40] B. Zhang, "Reliable classification of vehicle types based on cascade classifier ensembles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 1, pp. 322–332, 2013.

[41] "Malware Knowledge Base." `https://owl.nchc.org.tw/`.

[42] "Cuckoo Sandbox." `https://cuckoosandbox.org/`.

[43] Y.-H. Li, Y.-R. Tzeng, and F. Yu, "Viso: Characterizing malicious behaviors of virtual machines with unsupervised clustering," in *Cloud Computing Technology and Science (CloudCom), 2015 IEEE 7th International Conference on*, pp. 34–41, IEEE, 2015.

DOI:10.6814/THE.NCCU.MIS.001.2019.A05