Journal of Intelligent Learning Systems and Applications, 2015, 7, 42-57
Published Online May 2015 in SciRes. http://www.scirp.org/journal/jilsa
http://dx.doi.org/10.4236/jilsa.2015.72005

An Online Malicious Spam Email Detection System Using Resource Allocating Network with Locality Sensitive Hashing

Siti-Hajar-Aminah Ali[1], Seiichi Ozawa[1], Junji Nakazato[2], Tao Ban[2], Jumpei Shimamura[3]

[1]Graduate School of Engineering, Kobe University, Kobe, Japan
[2]National Institute of Information and Communications Technology (NICT), Tokyo, Japan
[3]Clwit Inc., Tokyo, Japan
Email: aminahh@uthm.edu.my, ozawasei@kobe-u.ac.jp, nakazato@nict.go.jp, bantao@nict.go.jp, shimamura@clwit.co.jp

Received 25 February 2015; accepted 20 April 2015; published 22 April 2015

Abstract

In this paper, we propose a new online system that can quickly detect malicious spam emails and adapt to the changes in the email contents and the Uniform Resource Locator (URL) links leading to malicious websites by updating the system daily. We introduce an autonomous function for a server to generate training examples, in which double-bounce emails are automatically collected and their class labels are given by a crawler-type software to analyze the website maliciousness called SPIKE. In general, since spammers use botnets to spread numerous malicious emails within a short time, such distributed spam emails often have the same or similar contents. Therefore, it is not necessary for all spam emails to be learned. To adapt to new malicious campaigns quickly, only new types of spam emails should be selected for learning and this can be realized by introducing an active learning scheme into a classifier model. For this purpose, we adopt Resource Allocating Network with Locality Sensitive Hashing (RAN-LSH) as a classifier model with a data selection function. In RAN-LSH, the same or similar spam emails that have already been learned are quickly searched for a hash table in Locally Sensitive Hashing (LSH), in which the matched similar emails located in "well-learned" are discarded without being used as training data. To analyze email contents, we adopt the Bag of Words (BoW) approach and generate feature vectors whose attributes are transformed based on the normalized term frequency-inverse document frequency (TF-IDF). We use a data set of double-bounce spam emails collected at National Institute of Information and Communications Technology (NICT) in Japan from March 1st, 2013 until May 10th, 2013 to evaluate the performance of the proposed system. The results confirm that the proposed spam email detection system has capability of detecting with high detection rate.

How to cite this paper: Ali, S.-H.-A., Ozawa, S., Nakazato, J., Ban, T. and Shimamura, J. (2015) An Online Malicious Spam Email Detection System Using Resource Allocating Network with Locality Sensitive Hashing. *Journal of Intelligent Learning Systems and Applications*, **7**, 42-57. http://dx.doi.org/10.4236/jilsa.2015.72005

Keywords

Malicious Spam Email Detection System, Incremental Learning, Resource Allocating Network, Locality Sensitive Hashing

1. Introduction

Emails have become one of the most frequently used methods for cyber attacks. The most worrying email-based attack is Targeted Malicious Email (TME) [1] [2]. In TME, attackers send malicious emails to certain people targeted in an organization, such as executives of large companies, high-ranking government personnel, military officials and even famous researchers, in order for the attackers to obtain valuable confidential information and latest research of the targeted people. In TME, an email often has an attachment with malicious codes that can be installed automatically upon opening without the victims realizing it. In some cases, the victims' computer will become the back door for the attackers who in turn have the authority to enter the network of the targeted persons and thus steal confidential information.

Another typical email-based cyber attack is the malicious spam email attack, which aims to spread numerous emails with Uniform Resource Locator (URL) links leading to malicious websites. Previously, malicious codes were sent through the attachment of such spam emails. However, many successful filters have been developed to detect malicious attachments. Thus, attackers are now turning to malicious spam campaigns that attack using the links attached in the emails. According to the Symantec annual report in 2014 [3], about 87 percent of scanned spam messages contained at least one URL hyperlink. Moreover, recent findings by Symantec [4] show a sharp rise of emails containing malicious links, from 7% in October 2014 to 41% in the following months. Apart from that, currently, attackers also use more relevant email contents [1] that are specific to their victims' line of work, besides addressing the name of the recipient in the email body to convince the victim that the email received is a normal email. For instance, a fake email notification regarding a conference or journal targeted towards a recipient with academic status, notifications regarding false documents such as telecommunication service bills, fax and voicemail in which the victims are given a link to get more information [4]. This technique is called Social Engineering [5], which Hadnagy [6] defines as "The Art of Human Hacking". It becomes difficult for normal users to distinguish not only between non-malicious and malicious spam emails but also spam email from normal emails.

The objective of this paper is to detect the malicious spam emails so that general users can be protected from being re-directed to malicious websites. For this purpose, we propose an autonomous online system for detecting malicious spam emails. In general, it is not easy to collect spam emails from individual persons because it is not usually permitted to access personal email spools. Therefore in the proposed system, we collect double-bounce spam emails that are delivered to unknown users. From the collected spam emails data, a classifier model is used to learn and classify the malicious spam emails. The updated connection weights of the classifier model are sent to a user's mailer software to improve the malicious spam email detection ability. Jungsuk [7] points out that the live period of malicious URLs is often very short, usually within a few days; thus, it is expected that introducing incremental learning to malicious spam email detection will be effective. The system can learn from the recent spam emails so that the spam email detection system is always up to date. On the other hand, spammers often use botnets to spread spam emails. For example, a botnet called Rustock which consists of approximately 1 million infected computers that networked together, is capable of sending up to 30 billion spam emails every day [8]. Since the distribution of such spam emails is done in a short time, we assume that the spam emails have the same or similar contents in general [9]. Hence, we adopt the Locality Sensitive Hashing (LSH) [10]-[14] to quickly select important training data to be learned. For this purpose, we adopt Resource Allocating Network with Locality Sensitive Hashing (RAN-LSH) as a classifier model in the proposed detection system. This model has the following two important properties: 1) the learning is carried out incrementally; and 2) only data within an untrained region are selected and learned even when a large amount of data is given.

This paper is organized as follows. Section 2 gives a brief explanation of RAN-LSH. The proposed system for detecting malicious spam emails is presented in Section 3. In Section 4, the performance of the spam email detection system is evaluated for a set of 20,448 double-bounce emails collected from 1st March, 2013 to 10th May, 2013. Finally, conclusions and future work are addressed in Section 5.

2. Resource Allocating Network with Locally Sensitive Hashing (RAN-LSH)[1]

Figure 1 demonstrates the overall architecture of the spam email detection system. In this section, we give a brief explanation of the RAN-LSH classifier [15] illustrated in **Figure 1(c)**. RAN-LSH is the extended model of the Resource Allocating Network (RAN) [16], where LSH is adopted to select essential training data and Radial Basis Function (RBF) bases for fast learning. There are three main components in RAN-LSH: hash table, data selection and classifier.

Algorithm 1 shows the overall learning procedures of RAN-LSH. During the initial learning phase (Lines 1 - 5), initial training data are used to obtain the most suitable values of the following two important parameters: RBF width σ and the number of partitions P. In addition, initial data are also used to obtain an initial hash table and initial structure of the classifier. After that, the incremental learning is carried out whenever training data are given to learn (Lines 7 - 17). In LSH, similar data are allocated in the same hash entry with a high probability. Therefore, the number of hash entries determines the granularity of input space representation, and too large number of hash entries would result in both high computational and memory costs in the data selection. Therefore, it is important to design the hash functions such that a suitable number of hash entries are created.

In RAN-LSH, we adopt Principal Component Analysis (PCA) to generate a proper number of hash functions by controlling the threshold of the accumulation ratio θ_a. Accumulation ratio $A(\boldsymbol{U}_l)$ is the ratio of input components in the approximated subspace over those in the whole input space [17]. Giving a proper value of θ_a based on a tolerant approximation error, a proper number of hash functions is automatically determined by selecting the number of partitions P via the cross-validation.

Let l be the subspace dimensions obtained by PCA. Then, the following linear transformation is considered to define hash values in LSH:

$$\boldsymbol{V} = \boldsymbol{U}_l^{\mathrm{T}} \boldsymbol{x} \tag{1}$$

where $\boldsymbol{V} = \{v_1, \cdots, v_l\}$, $\boldsymbol{U} = \{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_l\}$ and $\boldsymbol{x}$ are the l-dimensional projection vector, the matrix of l eigenvectors, an I-dimensional input vector, respectively. Each projection vectors $v_i\ (i = 1, \cdots, l)$ is then divi-

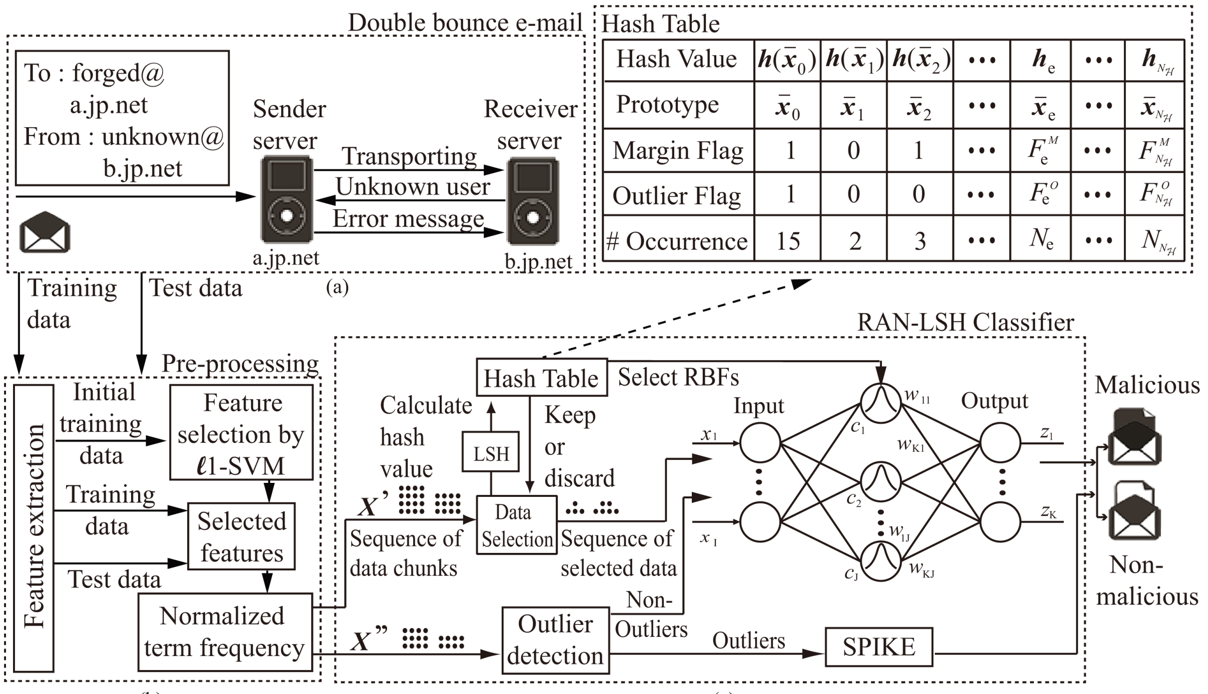


Figure 1. Network structure of the proposed autonomous malicious spam email detection system.

[1]This work has been submitted to a journal and currently under review.

Algorithm 1 RAN-LSH Learning Algorithm

```
Input: A set of $N_d$ training data $\boldsymbol{X}' = \{(\boldsymbol{x}_i, \boldsymbol{d}_i)\}_{i=1}^{N_d}$ and parameters: accumulation ratio $\theta_a$, output margin $\theta_m$ and
    tolerant distance $\theta_p$.
 1: Do the cross-validation for $N_0$ initial training data $\boldsymbol{X}_0 = \{(\boldsymbol{x}_i, \boldsymbol{d}_i)\}_{i=1}^{N_0}$ to determine the RBF width $\sigma^2$, the
    number of partitions $P$, output margin for outlier detection $\theta_o$ and occurrence frequency threshold $\theta_N$.
 2: Do PCA for $\boldsymbol{X}_0$, and obtain $l$ eigenvectors $\boldsymbol{U}_l$ such that the accumulation ratio is larger than $\theta_a$.
 3: Calculate projection vector $\boldsymbol{V}$ and for each $v \in \{v_1, \cdots, v_l\}$, find $v^+$ and $v^-$.
 4: Do *Update Hash Table* to create initial hash table where initial occurrence frequency $N_e$ of similar data and
    number of entry in a hash table $\mathcal{H}$ are set to 0.
 5: Do *RAN Learning* to construct initial RAN.
 6: loop
 7:   Input: A chunk of $N_d$ training data $\boldsymbol{X}' = \{(\boldsymbol{x}_i, \boldsymbol{d}_i)\}_{i=N_0+1}^{N_d}$.
 8:   Do *Update Hash Table* to get hash values $\boldsymbol{h}(\boldsymbol{x}_i)$.
 9:   Define $\hat{\boldsymbol{X}}' = \{(\boldsymbol{x}_i, \boldsymbol{d}_i, \boldsymbol{h}(\boldsymbol{x}_i))\}_{i=N_0+1}^{N_d}$.
10:   for all $\boldsymbol{x}_i \in \hat{\boldsymbol{X}}'$ do
11:     Find $\boldsymbol{h}(\boldsymbol{x}_i)$ in $\mathcal{H}$.
12:     if $F_e^M = 1$ for a matched entry then
13:       Remove $\boldsymbol{x}_i$ from $\hat{\boldsymbol{X}}'$ and $N_d \leftarrow N_d - 1$.
14:     end if
15:   end for
16:   Do *RAN Learning* using selected data $\hat{\boldsymbol{X}}'$.
17:   Do *Update Hash Table*.
18: end loop
```

ded into P partitions with equal size. θ_a controls the number of eigenvectors $\boldsymbol{U}_l$. To obtain a hash code, data is first projected on all the eigenvectors $\boldsymbol{u}_i$ and the hash code is obtained by combining the encoded values for the projections. When a large θ_a is adopted, the number of eigenvectors tends to become large and this would cause elongation of the length of a hash code.

As shown in **Algorithm 1** (see Lines 7 - 17), the incremental learning of RAN-LSH is carried out not only for RAN-LSH classifier but also for the hash table. Let us briefly explain the learning procedures in the following subsections.

2.1. Updating Hash Table

Algorithm 2 illustrates the steps to create and update the hash table which is used in RAN-LSH learning algorithm (Lines 4, 8 and 17 in **Algorithm 1**). Each subregion is allocated to an entry in a hash table, where each entry is composed of five items: hash value $\boldsymbol{h}(\overline{\boldsymbol{x}}_e)$, prototype $\overline{\boldsymbol{x}}_e$, margin flag F_e^M, outlier flag F_e^O and the occurrence frequency N_e (**Figure 1**: top right). The index or a hash value is used as a key to find a matched entry e of a similar item which has been registered previously in hash table (see the first condition in Line 6). Hash values $\boldsymbol{h}(\boldsymbol{x}) = \{H(v_1), \cdots, H(v_l)\}$ are a set of hash functions $H(v_i)$, which are given as follow:

$$H(v_i) = \max\left\{\left\lceil \frac{\min\left\{\max\left\{v_i, v_i^-\right\}, v_i^+\right\} - v_i^-}{v_i^+ - v_i^-} P \right\rceil, 1\right\} \tag{2}$$

where v_i^+ and v_i^- are the upper and lower values of projections v_i on the i th eigenvector $\boldsymbol{u}_i$, respectively. Here, P is the number of partitions which determines the granularity for a projection v_i.

The next item is prototype $\overline{\boldsymbol{x}}_e$. A prototype is the mean vector of all data allocated to each entry (Lines 7 and 11) which is calculated as follows:

$$\overline{\boldsymbol{x}}_e = \frac{\left(\overline{\boldsymbol{x}}_e' * N_e\right) + \sum_{i=1}^{n}\left(\tilde{\boldsymbol{x}}_i\right)}{N_e + n} \tag{3}$$

Algorithm 2 Update Hash Table

```
Input: Data $\boldsymbol{X}' = \{(\boldsymbol{x}_i, \boldsymbol{d}_i)\}_{i=1}^{N}$ and hash table $\mathcal{H} = \{\boldsymbol{h}_e, \bar{\boldsymbol{x}}_e, F_e^M, F_e^O, N_e\}_{e=1}^{N_\mathcal{H}}$.
Output: $\mathcal{H}$.
1:  Calculate the hash values $\boldsymbol{h}(\boldsymbol{x}_i)$ for all $\boldsymbol{x}_i \in \boldsymbol{X}'$ by Equations (1) - (2).
2:  Find $s(1, \cdots, S)$ subsets of unique hash values from $\{\boldsymbol{h}(\boldsymbol{x}_i)\}_{i=1}^{N}$ so that $\boldsymbol{h}_s \subset \{\boldsymbol{h}(\boldsymbol{x}_i)\}_{i=1}^{N}$.
3:  Sort all $\boldsymbol{x}_i \in \boldsymbol{X}'$ into the corresponding subset $\boldsymbol{h}_s$ where $\boldsymbol{h}(\boldsymbol{x}_i) = \boldsymbol{h}_s$.
4:  Define all data in each subset as $\tilde{\boldsymbol{X}}_s$.
5:  for $s = 1$ to $S$ do
6:      if $\boldsymbol{h}_s$ is registered in $\mathcal{H}$ (i.e., $\boldsymbol{h}_s = \boldsymbol{h}_e$) & $F_e^M = 1$ then
7:          Calculate the mean vector using Equation (3) as a prototype $\bar{\boldsymbol{x}}_e$ where $n = |\tilde{\boldsymbol{X}}_s|$.
8:          Set $N_e \leftarrow N_e + n$.
9:          break
10:     else
11:         Calculate the mean vector using Equation (3) as $\bar{\boldsymbol{x}}_e$.
12:         Calculate the output margin $\Delta z$ for $\bar{\boldsymbol{x}}_e$ by Equation (5) and set the margin flag $F_e^M$ and outlier flag $F_e^O$ by
            Equations (4) and (6) respectively.
13:         Set $N_e \leftarrow N_e + n$.
14:         Register $\{\boldsymbol{h}_e, \bar{\boldsymbol{x}}_e, F_e^M, F_e^O, N_e\}_{e=N_\mathcal{H}+1}$ in $\mathcal{H}$.
15:         Set $N_\mathcal{H} \leftarrow N_\mathcal{H} + 1$.
16:     end if
17: end for
```

where $\bar{\boldsymbol{x}}'_e$ and $\tilde{\boldsymbol{x}}_i$ are the previous prototype which is registered in the entry e and training data that belong to s th subset of unique hash values $\tilde{\boldsymbol{x}}_i \in \tilde{\boldsymbol{X}}_s$, respectively. Meanwhile, N_e is the previous occurrence frequency of the entry e and n is the number of training data that has a similar hash value. $\bar{\boldsymbol{x}}_e$ is regarded as the representative point of the subregion.

The third item is the margin flag F_e^M and it is calculated as follows (Line 12):

$$F_e^M = \begin{cases} 0, & (\Delta z(\bar{\boldsymbol{x}}_e) \le \theta_m) \\ 1, & (\text{otherwise}) \end{cases} \tag{4}$$

where the output margin Δz is given by subtracting the second largest network output $z_{k''}(\bar{\boldsymbol{x}}_e)$ from the largest network output $z_{k'}(\bar{\boldsymbol{x}}_e)$ as follows:

$$\Delta z(\bar{\boldsymbol{x}}_e) = \underset{k' \in \{1,\cdots,K\}}{\arg\max}\, z_{k'}(\bar{\boldsymbol{x}}_e) - \underset{k'' \in \{1,\cdots,K\} \setminus k'}{\arg\max}\, z_{k''}(\bar{\boldsymbol{x}}_e) \tag{5}$$

The fourth item is the outlier flag F_e^O which is determined as follows (Line 12):

$$F_e^O = \begin{cases} 0, & (F_e^M = 0) \\ 0, & (\theta_m \le \Delta z(\bar{\boldsymbol{x}}_e) < \theta_o \ \& \ N_e < \theta_N) \\ 1, & (\text{otherwise}) \end{cases} \tag{6}$$

The last item is the occurrence frequency N_e of similar data in an entry. Whenever a new training data is assigned to an entry, the occurrence frequency is increased by one. The details of margin flag F_e^M and outlier flag F_e^O are discussed in the following Section 2.2 and 3.4, respectively.

2.2. Data Selection and RBF Bases Selection Using LSH

When a large number of data are given simultaneously under an incremental learning environment, it is important to learn only essential data in a classifier model. Obviously, this is because the learning must be completed as quickly as possible; otherwise, the next data may be given before the learning is completed. In RAN-LSH, the data selection is conducted by using LSH. First, all N_d training data in a given training set are projected to l eigenvectors. Then, for each training data, the projection value is encoded into a hash code whose granularity is determined by the number of partitions P, and the obtained hash codes are transformed into a hash value.

If a matched entry with the same hash value is found and the margin flag F_e^M is "1", it means the classifier is well trained (Line 6 in **Algorithm 2**). Then, the mean vector is calculated and the occurrence frequency N_e is incremented by n. Otherwise (Line 10 in **Algorithm 2**), the output margin Δz in Equation (5) is calculated and the margin flag F_e^M and outlier flag F_e^O are updated by Equations (4) and (6), respectively. Note that the training data associated with the margin flag $F_e^M = 1$ are eliminated from the training set (see **Algorithm 1** Lines 12 - 14). On the contrary, if the margin flag F_e^M is "0", it means a given data should be trained. After the learning phase, the margin flag F_e^M would be updated. Nevertheless, updating the margin flag of every prototype in the hash table would increase the learning time. As mentioned before, the prototype with $F_e^M = 1$ means the classifier is "well-trained" around the prototype. Thus, this prototype does not need to be updated. Meanwhile, prototype with $F_e^M = 0$ should be updated because there would probably be regions that have become "well-trained" after the learning phase (Line 17 in **Algorithm 1** and Line 12 in **Algorithm 2**).

LSH is also used to find RBF bases near to the training data (Lines 8 - 13 in **Algorithm 3**). In RAN-LSH, only the connection weights connected to the selected RBF bases are updated in the following procedures. Firstly, the hash values of RBF bases $\boldsymbol{h}(\boldsymbol{c}_j)$ are retrieved from a hash table. Next, the LSH distance d_j for each j th RBFs is calculated as follows:

Algorithm 3 RAN Learning

Input: Data $\boldsymbol{X}' = \{(\boldsymbol{x}_i, \boldsymbol{d}_i, \boldsymbol{h}(\boldsymbol{x}_i))\}_{i=1}^{N}$, RAN, hash table $\mathcal{H}$ and parameter θ_p.
Output: RAN.

```
1:  for all $(\boldsymbol{x}_i, \boldsymbol{d}_i, \boldsymbol{h}(\boldsymbol{x}_i)) \in \boldsymbol{X}'$ do
2:    Calculate the outputs $\boldsymbol{z}$ for $\boldsymbol{x}_i$ and find the nearest center $\boldsymbol{c}^*$ to $\boldsymbol{x}_i$.
3:    Calculate the error: $E = \|\boldsymbol{z}(\boldsymbol{x}) - \mathbf{1}_c\|/K$
4:    if $E > \varepsilon$ & $\|\boldsymbol{x} - \boldsymbol{c}^*\| > \delta$ then
5:      Add an RBF unit (i.e., $J \leftarrow J + 1$) and set an RBF center, and the connection weights: $\boldsymbol{c}_J = \boldsymbol{x}_i$, and
        $\boldsymbol{w}_J = \boldsymbol{d}_i - \boldsymbol{z}$.
6:      Get an index set $\mathcal{C}$ of RBFs position in hash table as such $\boldsymbol{h}(\boldsymbol{c}_J)$ is the $e$th entry in the hash table $\mathcal{H}_{\{\mathcal{C}_J = e\}}$:
        $\boldsymbol{h}(\boldsymbol{c}_J) = \boldsymbol{h}(\bar{\boldsymbol{x}}_e)$.
7:    else
8:      for $j = 1$ to $J$ do
9:        Calculate LSH distance $d_j^*$ between hash values of RBF centers $\boldsymbol{h}(\boldsymbol{c}_j)$ given by $\boldsymbol{h}_{\mathcal{C}_j}$ and a training data
          $\boldsymbol{h}(\boldsymbol{x}_i)$ using Equation (7).
10:       if $d_j^* \le \theta_p$ then
11:         Define a set $\mathcal{R}$ of selected RBF centers (i.e $\mathcal{R} \leftarrow j$).
12:       end if
13:     end for
14:     for all $\{(\boldsymbol{c}_j, \boldsymbol{w}_j)\}_{j \in \mathcal{R}}$ do
15:       Calculate the outputs $\boldsymbol{\Phi}$ for selected RBF centers $\boldsymbol{c}_j$ $(j \in \mathcal{R})$ and a training data $\boldsymbol{x}_i$ by Equation (8).
16:       Decompose $\boldsymbol{\Phi}$ using SVD and obtain the matrices $\boldsymbol{V}, \boldsymbol{H}, \boldsymbol{U}$.
17:       Update connection weights $\boldsymbol{W}' = \boldsymbol{V}\boldsymbol{H}^{-1}\boldsymbol{U}^T\boldsymbol{D}$.
18:     end for
19:     Calculate the outputs $\boldsymbol{z}$ for $\boldsymbol{x}_i$.
20:     Calculate the error: $E = \|\boldsymbol{z}(\boldsymbol{x}) - \mathbf{1}_c\|/K$
21:     if $E > \varepsilon$ then
22:       Add a hidden unit and the connection weights: $\boldsymbol{c}_J = \boldsymbol{x}_i$, and $\boldsymbol{w}_J = \boldsymbol{d}_i - \boldsymbol{z}$.
23:       Get an index set $\mathcal{C}$: $\mathcal{C}_J = e$.
24:     end if
25:   end if
26: end for
```

$$d_j = \boldsymbol{h}(\boldsymbol{x}) - \boldsymbol{h}(\boldsymbol{c}_j) = \sum_{i=1}^{I} \left(\left| H(v_{x\_i}) - H(v_{cj\_i}) \right| \right) \tag{7}$$

Then, only RBF bases whose LSH distance is less than a threshold θ_p are selected for a learning purpose. This is because it is considered that if the LSH distance is large, the RBF output would become very small and the weight update could be negligible. Finally, the selected RBF bases are used to solve the linear equation in $\boldsymbol{\Phi W} = \boldsymbol{D}$ [18].

2.3. RAN Classifier

Let the number of inputs, RBF units, and outputs be I, J, and K, respectively. RBF outputs $\boldsymbol{y}(\boldsymbol{x}) = \{y_1(\boldsymbol{x}), \cdots, y_J(\boldsymbol{x})\}^{\mathrm{T}}$ and the network outputs $\boldsymbol{z}(\boldsymbol{x}) = \{z_1(\boldsymbol{x}), \cdots, z_K(\boldsymbol{x})\}^{\mathrm{T}}$ of inputs $\boldsymbol{x} = \{x_1, \cdots, x_I\}^{\mathrm{T}}$ are calculated as follows:

$$y_j(\boldsymbol{x}) = \exp\left(-\frac{\left\| \boldsymbol{x} - \boldsymbol{c}_j \right\|^2}{\sigma_j^2} \right), \quad (j = 1, \cdots, J) \tag{8}$$

$$z_k(\boldsymbol{x}) = \sum_{j=1}^{J} w_{kj} y_j(\boldsymbol{x}) + \xi_k, \quad (k = 1, \cdots, K) \tag{9}$$

where $\boldsymbol{c}_j = \{c_{j1}, \cdots, c_{jI}\}^{\mathrm{T}}$, σ_j^2, w_{kj} and ξ_k are the center of j th RBF unit, the variance of the j th RBF unit, the connection weight from the j th RBF unit to the k th RBF unit and the bias, respectively.

Algorithm 3 shows the learning algorithm of RAN classifier. In RAN-LSH, RBF centers are not trained but selected based on the output error. If the output error is large, it indicates that a new RBF unit should be added (Lines 5 and 22). As mentioned above, only connection weights for active RBF units are updated (Lines 15 - 17).

3. The Proposed Malicious Spam Email Detection System

Figure 1 illustrates the architecture of the proposed autonomous online malicious spam email detection system which is composed of three components: 1) autonomous spam email collection system; 2) text processing and feature transformation; and 3) RAN-LSH classifier embedded with the data selection and outlier detection mechanisms.

As mentioned in Section 2.2, learning all the given data is not a good strategy under incremental learning environments because the learning may not be completed before a new data set is given [19]. To enhance the adaptibility to dynamic environments, the learning should be carried out with essential data that are selected in an online fashion. There are two types of essential data for a learning purpose. The first type is the data located close to a class boundary [20], while the other is the data located outside of the learned region (*i.e.*, outlier). In order to ensure fast and accurate learning, the data selection mechanism should be introduced into a classifier model to find such essential data from a given chunk of data.

The first type of essential data has been discussed in Section 2.2. On the other hand, the second type of essential data are selected by the outlier detection. This type of essential data selection is introduced into the previous RAN-LSH classifier. The outlier detection relies on the output margin and the number of occurrence of similar data in the input space which are represented by outlier flag F_e^O.

In the following subsection, we explain the details of the three components of the autonomous online malicious spam email detection system, as well as the autonomous labeling system.

3.1. Autonomous Spam Email Collection System

Figure 1(a) illustrates the process of obtaining double bounce emails. Let us consider a case that a spammer sends a large number of emails. In many cases, almost all emails will reach existing users. However, it is very likely that some email addresses are no longer in use for some reason. Therefore, the email server would return such emails with unknown addresses to the sender. If the spammer intends to send a malicious email, it is also likely that the spammer has faked the originating address and such emails would be re-sent to the receiver. This type of unreachable error email is called "double-bounce email" [7] and they are usually disposed of by the email server on the receiver's side. We utilize this mechanism of generating double-bounce emails to collect malicious spam emails automatically.

3.2. Autonomous Labeling System

To use double-bounce emails as training data under the supervised learning, we would need their class labels. Needless to say, spammers try to conceal their malicious intention; therefore, it is not easy to determine the maliciousness from the collected double-bounce emails. The only way to identify the maliciousness is by clicking the URLs. Evidently, this is very dangerous for general users; therefore, we use a crawling-type web maliciousness analyzer called SPIKE, which was developed by the National Institute of Information and Communications Technology (NICT) in Japan.

Figure 2 illustrates how the maliciousness of URLs in a spam email is analyzed in SPIKE [21]. The URL links in the email are first extracted from a double-bounce email and SPIKE downloads the html file and attached materials (e.g., java scripts, pdf, doc files) in the entrance page. It then continues to find other URLs in the downloaded pages again. This process is conducted recursively by crawling the linked websites, and all the downloaded materials are analyzed. Emails that are only link to a normal webpage with non-malicious contents are considered as non-malicious spam emails (*i.e.*, all contents of Webpage 1 - 6 in **Figure 2** are normal), whereas the emails with at least one suspicious content (*i.e.*, one of Webpage 1 - 6 in **Figure 2** is malicious) are identified as malicious spam emails.

3.3. Text Processing and Feature Transformation

In order for the classifier to carry out the classification task effectively, the classifier requires instances as the input instead of the raw spam emails for the learning purpose. The instances consist of informative features with a fixed-length which are extracted from the emails. Thus, appropriate pre-processing steps are required so that the arbitrary data of text messages are transformed into features with numerical features. **Figure 1(b)** demonstrates the pre-processing module of the spam email detection system. Feature extraction of spam emails involves tokenizing and lemmatizing the documents into bag-of-words (BoW). Tokenization breaks the sentences in the emails into pieces of words and removes frequent words called *stop words* such as "the", "which", "are", etc. Besides filtering out *stop words*, lemmatization also reduces the number of words in BoW by transforming redundant words that end with "*ing*", "*ed*" and "*s*" into their root word (e.g., "learned" to "learn").

The BoW features usually consist thousands or millions of feature vectors. In general, only some features are informative and are able to differentiate different classes. Therefore, feature selection is carried out to select the most informative features in order to reduce the number of dimensions and avoid the computational complexity. Firstly, the initial training data are transformed into feature vectors with term frequency-inverse document fre-

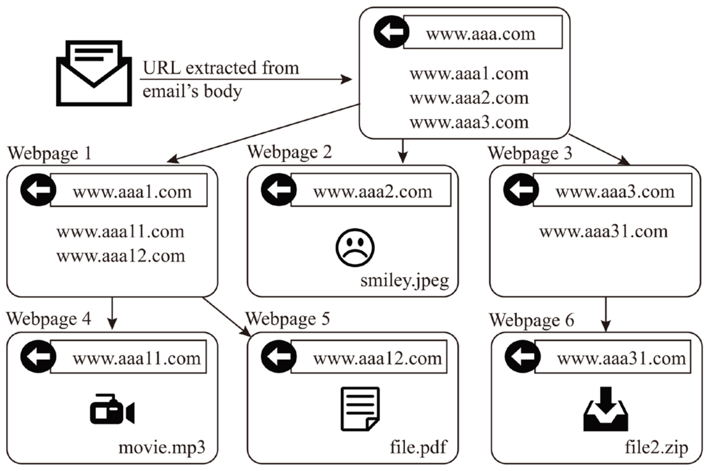


Figure 2. Example of web crawler and content analysis using SPIKE.

quency (TF-IDF) feature representation. Next, linear ℓ_1 Support Vector Machines (SVM) is used as a feature selection strategy that requires two steps; training using linear SVM [22] and eliminating features with low weights [23]. SVM is able to find a decision boundary by optimizing the following objective function:

$$\begin{aligned} &\min_{w,b} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{m}\xi_i \\ \text{s.t.}\quad & y_i\left(\boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}_i)+b\right) \le 1-\xi_i, \quad i=1,\cdots,m \\ & \xi_i \le 0, \quad i=1,\cdots,m. \end{aligned} \tag{10}$$

which maximizes the margin $\|\boldsymbol{w}\|^2$ of hyperplane $\boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}_i)+b$ between two classes $y_i \in \pm 1$ and contains only a minimum training error ξ_i (*i.e.*, training data located above the support vectors which belong to the class of the training data). The parameter C controls the trade off between margin maximization and errors of the SVM on training data, where a larger C corresponds to a higher penalty to errors. The weights $\boldsymbol{w}$ obtained is used to select N_f number of features by choosing the highest N_f -rank weights.

To represent the selected features of initial training data and the remaining training data, the normalized TF-IDF [24] is used to measure the importance of a word to a document (*i.e.*, document refers to the spam email) in the collection of documents given by the following equation:

$$\text{TF-IDF}(t,d,D) = \frac{f_d(t)}{\sum_t f_d(t)} \times \log\left(\frac{|D|}{1+|\{d \in D : t \in d\}|}\right) \tag{11}$$

where $f_d(t)$, $\sum_t f_d(t)$, $|D|$ and $|\{d \in D : t \in d\}|$ are the frequency of term t in a document d, the total frequencies of all terms in document d, the total number of document in corpus and number of documents which have term t, respectively. The normalized term frequency (TF) is used to provide a balanced value to all documents that have a different number of words. If term t appears frequently in a document d_i and seldom occurs in other documents in D, the value of TF-IDF would be high where both TF and inverse document frequency (IDF) obtain high values. This indicates that term t is important to document d_i. Otherwise, if either the occurrence of term t is low in document d_i or term t is always appears in other documents in D, this would indicate that term t is not important to document d_i where the value of TF-IDF is low or "0".

After going through the entire procedure above, these data are used as the input to the classifier model. The details of the classifier model are discussed in the previous Section 2.2.

3.4. Outlier Detection

Although SPIKE can judge the maliciousness of spam emails, the analysis takes time, from a few minutes to even longer than ten minutes. Therefore, it is difficult to check all the collected double-bounce emails by SPIKE in real time. We introduce the outlier detection mechanism into RAN-LSH in order to reduce the number of spam emails to be checked by SPIKE. That is, only a new type of unknown spam emails (*i.e.*, outlier) should be selected and sent to SPIKE for labeling. For this purpose, we propose a spam email detection system by combining RAN-LSH classifier [15] and SPIKE, so that the learning time is accelerated compared to when using SPIKE alone. In this study, we detect an outlier based on the output margin θ_m, outlier threshold θ_o and the occurrence frequency threshold θ_N. The data with low output margins are considered as unknown emails for the current classifier and thus should be categorized as outlier. In addition, the number of similar data in each entry N_e is also important to decide whether the data is outlier or not. We assume that the data that do not frequently occurred (*i.e.*, data allocated to an entry with small N_e) can also be categorized as outlier although the output margins are slightly higher than θ_m. The outlier flag F_e^O is calculated using Equation (6). The algorithm of the outlier detection is summarized in **Algorithm 4**.

4. Performance Evaluation

4.1. Experimental Setup

The detection performance is evaluated under incremental learning settings to study the following effects: 1) the effect of threshold parameters and 2) the effectiveness of daily updates.

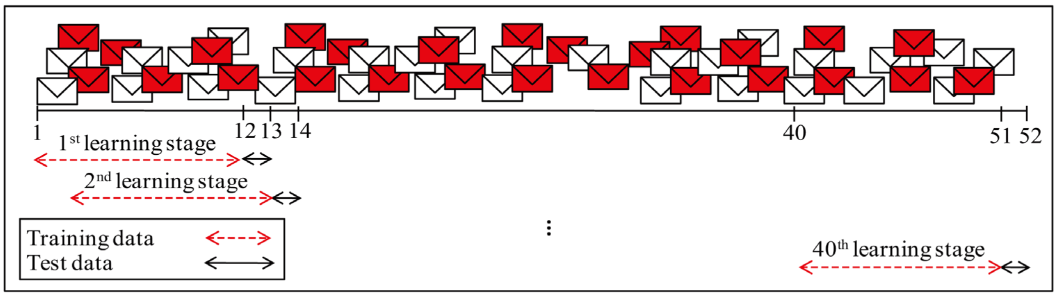
Algorithm 4 Outlier Detection

```
Input: Test data $\boldsymbol{X}'' = \{\boldsymbol{x}_i\}_{i=1}^{N}$, hash table $\mathcal{H}$ and SPIKE.
Output: Class label $\boldsymbol{d}_i$
1: loop
2:    Calculate the hash value $\boldsymbol{h}(\boldsymbol{x}_i)$ by Equations (1) - (2).
3:    Find $\boldsymbol{h}(\boldsymbol{x}_i)$ in $\mathcal{H}$.
4:    if $F_e^O = 0$ then
5:       Assign the data as outliers.
6:       Get class label $\boldsymbol{d}_i$ using SPIKE.
7:    else
8:       Get class label $\boldsymbol{d}_i$ using the spam email detection system.
9:    end if
10: end loop
```

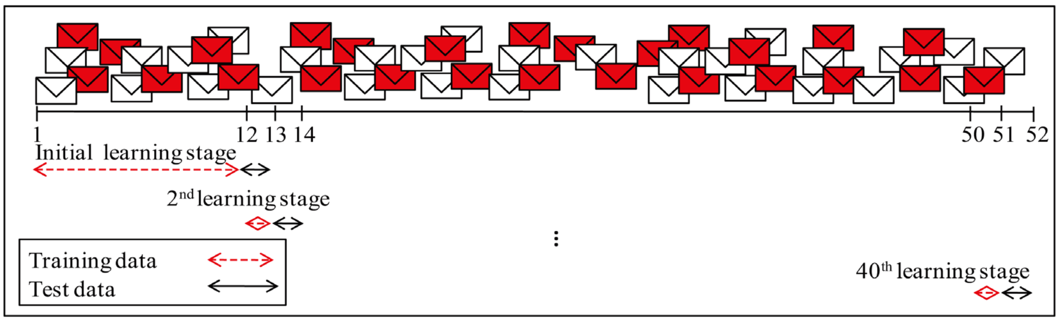
In the former experiment, we investigate the effects of the following three threshold parameters to the performance: threshold of accumulation ratio θ_a, that of output margin θ_m, and that of tolerant distance θ_p. In the latter experiment, we study the effect of incremental learning through comparison with the batch learning scheme. **Figure 3** illustrates how labeled spam emails are trained in (a) batch learning scheme and (b) incremental learning scheme. In the batch learning scheme, we adopt the conventional RBF network (RBFN) (*i.e.*, RBFN usually used as batch learning [25]) as a classifier and a sliding window is introduced to define a data set to be trained every day. In this experiment, the time-window size is preliminarily determined as 12 days via the cross-validation using the spam emails collected during a different period. Therefore, as seen in **Figure 3(a)**, the first learning stage is carried out on Day 12 using a set of spam emails collected from Day 1 to Day 12, and the data set from Day 13 is used to test the performance. Then, the time-window is shifted by one day at the second learning stage; that is, a set of spam emails collected from Day 2 to Day 13 is used for training, and the data set from Day 14 is used to test. Note that RBFN is retrained with a set of 12-day spam emails at every learning stage in a batch mode. On the other hand, in the incremental learning scheme, batch learning is first applied to an initial data set, which is composed of spam emails collected in the first 12 days. After that, this initial detection model is updated daily using one day training data and the system is tested using the next day data. A set of spam emails collected is forwarded to SPIKE to get the labels on maliciousness, and the pairs of a spam email and its class label are used as a training set on the next day. The number of collected spam emails is different every day. Their maximum, minimum, and average numbers are 756, 26, and 207, respectively.

In this detection system, three parameters are determined empirically. The parameters are accumulation ratio θ_a, output margin θ_m and tolerant distance θ_p. In the following experiment, the parameters are set to: $\theta_a = 0.9$, $\theta_m = 0.2$, and $\theta_p = 2$. Meanwhile, the error threshold ε is set to 0.5. The other parameters, RBF width σ, the number of partitions P, errors penalty of SVM C, outlier threshold θ_o, occurrence frequency threshold θ_N and time-window size are determined in the initial learning phase through the cross-validation. There are 20,448 double bounce emails with 8334 malicious spam emails and 12,114 non-malicious spam emails used in this study that were collected from 1st March 2013 to 10th May 2013.

It is important for the proposed system to correctly classify not only positive examples (malicious spam emails) but also negative ones (non-malicious spam emails). Therefore, in this study, three evaluations are used which are: recall rate, precision rate and F1 measure rate. The actual class labels are the class labels given by SPIKE, whilst the prediction labels are obtained by the detection system. The recall rate and the precision rate measure the ability of the detection system to classify the malicious spam emails (positive samples) correctly which takes into account different types of error. On the other hand, the F1 measure is the harmonic mean of recall and precision rate. The recall rate considers type II error (*i.e.*, a malicious spam email is wrongly classified as non-malicious spam email), whereas the precision rate considers type I error (*i.e.*, a non-malicious spam email is wrongly classified as malicious spam email). If the malicious spam email detection system obtained a low recall rate, the users are exposed to the danger of malware infection because some users may click an URL that leads to malicious websites. Such a misclassification must be avoided in any cases. In the second situation, there is low risk of the malware attack. The system only gives strict conditions where most of the non-malicious spam emails are categorized as malicious spam emails. Therefore, to design a good malicious spam email detection system, it is crucial to have at least high percentage of recall rate to reduce the risk of malware attack.



(a)



(b)

Figure 3. Comparison of learning scheme between batch learning and incremental learning. (a) Batch learning; (b) Incremental learning.

However, if the malicious spam email detection system is able to obtain high rate of both recall and precision, we can say that the developed system is nearly a perfect detection system.

4.2. Effects of Threshold Parameters

First, let us examine the threshold parameters and their effect to the detection system. Here, we study the influence of θ_a, θ_m and θ_p so that the parameters are optimized to ensure fast learning property of the detection system while having low misclassification rate. The first parameter is the threshold of accumulation ratio θ_a. If θ_a is set to a large value, an eigenspace to define hash functions has high-dimensions. Thus, the length of a hash code becomes long, resulting in the enlargement of a hash table. Therefore the searching of similar data would require a longer time since there are many hash values registered in hash table. The next parameter is the output margin threshold θ_m. This parameter controls the amount of selected data to be learned by RAN-LSH. As the value of θ_m is set to a higher value, the representation of the "not well-learned" region would become wider and the number of selected data is increased in the incremental learning phase, resulting in slower learning. The third important parameter is the tolerant distance θ_p which determines the distance of near RBF bases. By updating the weights of only near RBF bases (*i.e.*, using small value of θ_p), the time needed to solve the linear equation using Singular Value Decomposition (SVD) is shorten, thus the learning time would be accelerated.

To determine an appropriate value of each parameter, the cross-validation is performed for the initial training set, and the obtained parameter values are fixed over the incremental phase. **Table 1(a)** and **Table 1(b)** show the F1 measure and the learning time, respectively, using several combination values of the accumulation ratio θ_a and output margin θ_m. As seen in **Table 1(a)**, the highest F1 measure is obtained when $\theta_a = 0.9$ and $\theta_m = 0.2$. For output margin $\theta_m \le 0.2$, F1 measure does not differ much from the F1 measure with output margin $\theta_m = 0.2$. This result is not surprising because high value of output margins represents that the data are "well-learned" or correctly classified. By adding these data, it is expected that the classification rate would not improve although the number of selected data and RBF bases created are increased. As a result for learning using more training data shown by a higher θ_m, the learning time would also increase. We assume that the data which are located on the border of different classes should have network outputs in the range of "0.4" to "0.6".

Table 1. The evaluation using several values of accumulation ratio θ_a and output margin threshold θ_m for the spam email detection system. The performance measures are: (a) the F1 measure [%], and (b) initial learning time [sec.].

(a)

θ_a	θ_m				
	0.05	0.1	0.2	0.3	0.4
0.5	82.8	87.0	88.4	88.2	88.1
0.7	86.6	87.3	90.5	87.8	88.2
0.9	87.0	87.7	90.8	88.0	87.9

(b)

θ_a	θ_m				
	0.05	0.1	0.2	0.3	0.4
0.5	126.5	128.6	130.1	130.3	131.2
0.7	128.0	129.7	130.3	130.9	132.6
0.9	128.6	129.8	132.1	132.7	139.8

Therefore, the output margins θ_m can be estimated to be between “0” to “0.2” where the data with output margins in these range are important to be learned to reduce the misclassification rate.

On the other hand, **Table 2** demonstrates the suitable value of the tolerant distance θ_p using appropriate value of θ_a and θ_m obtained previously which are “0.9” and “0.2”, respectively. Tolerant distance θ_p also give influence to the classification rate and the speed of the detection system by controlling the distance which defines the area of near RBF bases. As we can see, the suitable value for the tolerant distance is 2. It means that only the RBF centers that differ from the given training data at two projection vector v_i are used to update the weight. If the θ_p is too small, it indicates that the area of selected RBF bases is not enough to approximate the weights correctly. Whereas for θ_p that is too large, it would be similar to the approach of updating weights using all RBF bases. Thus, the size of RBF outputs $\mathbf{\Phi}$ in $\mathbf{\Phi W} = \mathbf{D}$ would be bigger and therefore, the decomposition steps using SVD would require a longer time. Even though the results show the evaluation performance during initial learning, we expect a similar result from the incremental learning phase. It is because parameter θ_a, θ_m and θ_p are also required during the incremental learning phase. For the next experiment, we set the value of θ_a, θ_m and θ_p to be “0.2”, “0.9” and “2”, respectively.

4.3. Effectiveness of Incremental Learning

All learning parts in the detection system including pre-processing and classifier module are very crucial which give effect to the performance result. In this experiment, we compare the performance of the proposed online detection system with different learning scheme and classifier model to see the competency of the proposed method. **Figure 4** and **Figure 5** show the recall rate and precision rate for the detection system with the following three combinations of classifiers and learning schemes: RBFN (batch learning), RAN (incremental learning), and RAN-LSH (incremental learning) (see **Figure 3**). The batch learning is carried out using 12-days of training data and it is retrained incrementally. While for the incremental learning, the classifier is updated incrementally using 1-day of training data. As seen in **Figure 4** and **Figure 5**, the proposed one-pass learning of the detection system is capable to learn and carry out the classification task effectively since our proposed system obtained almost the same classification rate as the memory-based learning approach (*i.e.*, batch learning). In fact, our proposed method does not need large memory size to store the training data compared to the memory-based learning. In this study, 12-days length of window size is used for the batch learning to learn incrementally, whereas for incremental learning, only 1-day data set is used as training data. Besides that, we also compare the performance of conventional classifier RAN using the same incremental learning scheme. Our previous study in [15] shows that RAN-LSH can learn fast. As seen in **Figure 4** and **Figure 5**, our detection system obtains a comparable result against the conventional classifier model.

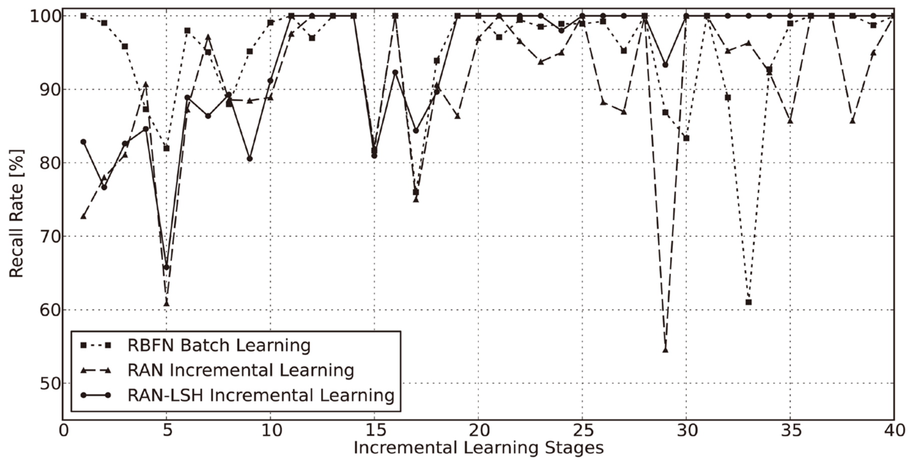


Figure 4. Transitions of recall rates in the malicious spam email detection system with three learning schemes.

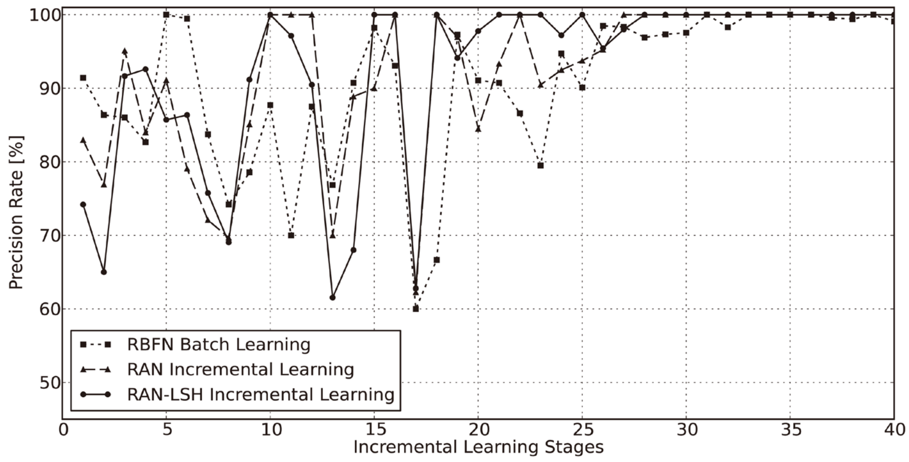


Figure 5. Transitions of precision in the malicious spam email detection system with three learning schemes.

Table 2. The performance using different values of tolerant distance θ_p.

Evaluation	θ_p			
	0	1	2	5
F1 measure [%]	88.5	88.8	89.3	89.3
Initial learning time [sec.]	138.2	139.1	140.1	144.9

4.4. Overall Performance of Malicious Spam Email Detection System

The overall performance is evaluated by averaging over the whole incremental learning phase. The recall rate, precision rate and F1-measure for the three learning models are summarized in **Table 3**. From **Table 3**, we can see that the proposed detection system can learn 482 times faster than the conventional RBFN model, and can learn 46 times faster than RAN. This is because our detection system can find a set of similar data in a given training set very quickly using LSH; thus, only a data set falling in an untrained region are selected to learn and the others are discarded. In addition, only near RBF centers are used to update the connection weights. The recall rate and F1 measure of RBFN are higher than those of the other incremental models because this model keeps a large number of data for training; where this requires high computational costs and large memory. Since

Table 3. Overall performance of malicious spam email detection system.

Performance evaluation	RBFN batch	RAN incremental	RAN-LSH incremental
Recall Rate [%]	94.6	89.4	90.7
Precision Rate [%]	90.7	90.8	87.2
F1 Measure [%]	92.2	89.3	87.7
Learning Time [sec.]	4532.7	432.0	9.4

the classification rate of recall rate, precision rate and F1 measure do not differ much from the other model, we can conclude that the proposed system is able to update efficiently and able to give class label of the incoming emails within a short time.

5. Conclusions

We have proposed a malicious spam email detection system using BoW features, where the classifier adopts LSH to select essential data and near RBF bases. We use two types of essential data: 1) the data located close to a class boundary; and 2) the data located outside of the learned region (*i.e.*, outlier). The proposed scheme provides desirable learning characteristics as an autonomous malicious spam email detection system and able to adapt to new trends of malicious emails quickly. In addition, our detection system is quite fast compared with SPIKE which often needs a long time to complete the maliciousness analysis. By using the proposed system, it is possible to give proper alerts to users quickly based on up to date information. Since the learning is quite fast and the detection performance is comparable to the conventional models, we can conclude that the proposed system is suitable to be implemented in an email client software on the user side.

Currently, the proposed detection system has no pruning function for RBF bases. Therefore, as the learning is continued for a long time, the number of RBF could be increased excessively, and this causes longer learning time. Then, in the worst scenario, the learning may not converge before new training data are given. To avoid such a disastrous situation, a proper number of RBF bases should always be maintained by introducing an online pruning mechanism into RAN-LSH. Besides that, our detection system uses selected features from initial learning training data. As our future work, we intend to construct an adaptive hash table to adapt to the changes of feature vectors from the recent BoW without forgetting the previous knowledge. It is expected that the detection system would be more stable and robust to the new malicious spam email attacks.

Acknowledgements

This work is partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research (C) 24500173, the University of Tun Hussein Onn Malaysia (UTHM) and the Ministry of Education Malaysia (KPM).

References

[1] Vuong, T.P. and Gan, D. (2012) A Targeted Malicious Email (TME) Attack Tool. 6*th International Conference on Cybercrime*, *Forensics*, *Education and Training* (*CFET*), Christ Church Canterbury.

[2] Nagarjuna, B.V.R.R. and Sujatha, V. (2013) An Innovative Approach for Detecting Targeted Malicious E-Mail. *International Journal of Application or Innovation in Engineering & Management* (*IJAIEM*), **2**, 422-428.

[3] Symantec Corporation (2014) Internet Security Threat Report 2014, Vol. 19, 1-98.
http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v19_21291018.en-us.pdf

[4] Hurcombe, J. (2014) Malicious Links: Spammers Change Malware Delivery Tactics.
http://www.symantec.com/connect/blogs/malicious-links-spammers-change-malware-delivery-tactics

[5] Amin, R.M. (2011) Detecting Targeted Malicious Email through Supervised Classification of Persistent Threat and Recipient Oriented Features. Ph.D. Dissertation, Dept. Eng. and Applied Sciences, George Washington University, Washington.
http://www.researchgate.net/publication/224265677_Detecting_Targeted_Malicious_Email_Using_Persistent_Threat_and_Recipient_Oriented_Features

[6] Hadnagy, C. (2011) Social Engineering: The Art of Human Hacking. Wiley, Indianapolis.

[7] Jungsuk, S. (2011) Clustering and Feature Selection Methods for Analyzing Spam Based Attacks. *Journal of the National Institute of Information and Communications Technology*, **58**, 35-50.

[8] Criddle, L. What Are Bots, Botnets and Zombies? http://www.webroot.com/za/en/home/resources/tips/pc-security/security-what-are-bots-botnets-and-zombies

[9] Nazirova, S. (2011) Survey on Spam Filtering Techniques. *Communications and Network*, **3**, 153-160. http://www.scirp.org/journal/PaperInformation.aspx?PaperID=6769#.VPkYAzWlilN http://dx.doi.org/10.4236/cn.2011.33019

[10] Datar, M., Immorlica, N., Indyk, P. and Mirrokni, V.S. (2004) Locality-Sensitive Hashing Scheme Based on p-Stable Distributions. *Proceedings of Symposium on Computational Geometry* (*SoCG'04*), 253-262. http://dl.acm.org/citation.cfm?id=997857 http://dx.doi.org/10.1145/997817.997857

[11] Andoni, A. and Indyk, P. (2008) Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. *Communications of the ACM*, **51**, 117-122. http://dl.acm.org/citation.cfm?id=1327494 http://dx.doi.org/10.1145/1327452.1327494

[12] Gu, X., Zhang, Y., Zhang, L., Zhang, D. and Li, J. (2013) An Improved Method of Locality Sensitive Hashing for Indexing Large-Scale and High-Dimensional Features. *Signal Processing*, **93**, 2244-2255. http://dl.acm.org/citation.cfm?id=2464367 http://dx.doi.org/10.1016/j.sigpro.2012.07.014

[13] Lee, K.M. and Lee, K.M. (2012) Similar Pair Identification Using Locality-Sensitive Hashing Technique. *Proceedings of Joint* 6*th International Conference on Soft Computing and Intelligent Systems* (*SCIS*) *and* 13*th International Symposium on Advanced Intelligent Systems* (*ISIS*), 2117-2119. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=6505385 http://dx.doi.org/10.1109/SCIS-ISIS.2012.6505385

[14] Shen, H., Li, T., Li, Z. and Ching, F. (2008) Locality Sensitive Hashing Based Searching Scheme for a Massive Database. *Proceedings of IEEE Southeastcon*'08, 123-128. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4494271

[15] Ali, S.H.A., Fukase, K. and Ozawa, S. (2013) A Neural Network Model for Large-Scale Stream Data Learning Using Locally Sensitive Hashing. *Neural Information Processing Lecture Notes in Computer Science*, 369-376. http://link.springer.com/chapter/10.1007%2F978-3-642-42054-2_46

[16] Platt, J. (1991) A Resource-Allocating Network for Function Interpolation. *Neural Computation*, **3**, 213-225. http://sci2s.ugr.es/keel/pdf/algorithm/articulo/plat1991.pdf http://dx.doi.org/10.1162/neco.1991.3.2.213

[17] Ozawa, S., Pang, S. and Kasabov, N. (2008) Incremental Learning of Chunk Data for Online Pattern Classification Systems. *IEEE Transactions on Neural Networks*, **19**, 1061-1074. http://www.lib.kobe-u.ac.jp/repository/90001005.pdf http://dx.doi.org/10.1109/TNN.2007.2000059

[18] Haykin, S. (1999) Neural Networks: A Comprehensive Foundation. Prentice Hall, Upper Saddle River.

[19] Langley, P. (1994) Selection of Relevant Features in Machine Learning. *Proceedings of the AAAI Fall Symposium on Relevance*, New Orleans, 4-6 November 1994, 140-144.

[20] Oyang, Y.J., Hwang, S.C., Ou, Y.Y., Chen, C.Y. and Chen, Z.W. (2005) Data Classification with Radial Basis Function Networks Based on a Novel Kernel Density Estimation Algorithm. *IEEE Transactions on Neural Networks*, **16**, 225-236. http://dx.doi.org/10.1109/TNN.2004.836229 http://ieeexplore.ieee.org/xpl/abstractAuthors.jsp?tp=&arnumber=1388471&url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel5%2F72%2F30214%2F01388471.pdf%3Farnumber%3D1388471

[21] Dai, Y., Tada, S., Ban, T., Nakazato, J., Shimamura, J. and Ozawa, S. (2014) Detecting Malicious Spam Mails: An Online Machine Learning Approach. *Neural Information Processing Lecture Notes in Computer Science*, **8836**, 365-372. http://link.springer.com/chapter/10.1007%2F978-3-319-12643-2_45

[22] Cortes, C. and Vapnik, V. (1995) Support-Vector Networks. *Machine Learning*, **20**, 273-297. http://link.springer.com/article/10.1023%2FA%3A1022627411411 http://dx.doi.org/10.1007/BF00994018

[23] Brank, J., Grobelnik, M., Milić-Frayling, N. and Mladenić, D. (2002) Feature Selection Using Linear Support Vector Machines. *Proceedings of the* 3rd *International Conference on Data Mining Methods and Databases for Engineering*, *Finance*, *and Other Fields*, Bologna, Italy, 25-27 September 2002, 84-89.

[24] Robertson, S.E., Walker, S., Jones, S., Hancock-Beaulieu, M.M. and Gatford, M. (1996) Okapi at TREC-3. *Proceedings of the Third NIST Text Retrieval Conference* (*TREC*3), NIST Special Publication 500-225, Washington DC, 109-126.

[25] Ozawa, S., Tabuchi, T., Nakasaka, S. and Roy, A. (2010) An Autonomous Incremental Learning Algorithm for Radial Basis Function Networks. *Journal of Intelligent Learning Systems and Applications*, **2**, 179-189.
http://www.scirp.org/journal/PaperInformation.aspx?PaperID=3333#.VPkOYTWlilM
http://dx.doi.org/10.4236/jilsa.2010.24021