Neurocomputing 275 (2018) 2401–2413

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

A LSTM based framework for handling multiclass imbalance in DGA botnet detection

Duc Tran*, Hieu Mac, Van Tong, Hai Anh Tran, Linh Giang Nguyen

Bach Khoa Cybersecurity Centre, Hanoi University of Science and Technology, No. 1, Dai Co Viet Road, Hanoi, Viet Nam

ARTICLE INFO

Article history:
Received 26 May 2017
Revised 22 October 2017
Accepted 6 November 2017
Available online 16 November 2017

Communicated by Dr. Xin Luo

Keywords:
Multiclass imbalance
Long Short-Term Memory Networks
LSTM
Cost-sensitive learning

ABSTRACT

In recent years, botnets have become a major threat on the Internet. Most sophisticated bots use Domain Generation Algorithms (DGA) to pseudo-randomly generate a large number of domains and select a subset in order to communicate with Command and Control (C&C) server. The basic aim is to avoid blacklisting, sinkholing and evade the security systems. Long Short-Term Memory network (LSTM) provides a mean to combat this botnet type. It operates on raw domains and is amenable to immediate applications. LSTM is however prone to multiclass imbalance problem, which becomes even more significant in DGA malware detection. This is due the fact that many DGA classes have a very little support in the training dataset. This paper presents a novel LSTM.MI algorithm to combine both binary and multiclass classification models, where the original LSTM is adapted to be cost-sensitive. The cost items are introduced into backpropagation learning procedure to take into account the identification importance among classes. Experiments are carried out on a real-world collected dataset. They demonstrate that LSTM.MI provides an improvement of at least 7% in terms of macro-averaging recall and precision as compared to the original LSTM and other state-of-the-art cost-sensitive methods. It is also able to preserve the high accuracy on non-DGA generated class (0.9849 F1-score), while helping recognize 5 additional bot families.

1. Introduction

Botnet is a collection of malware-infected machines or bots. It has become the main mean for cyber-criminals to send spam mails, steal personal data and launch distributed denial of service attacks [1]. Most bots today rely on domain generation algorithm (DGA) to generate a list of candidate domain names in the attempt to connect with the so-called command and control (C&C) server. This algorithm is also known as domain fluxing, where the domain list is changed over the time to avoid the limitations that allow researchers to shut down botnets [2].

Uncovering DGA is critical in security community. Existing solutions are largely based on reverse engineering and blacklisting the C&C domains to detect bots and block their traffic. Reverse engineering requires a malware sample that may not be always possible in practice [1,2]. Blacklisting, on the other hand becomes increasingly difficult as the rate of dynamically generated domains increases [3]. Kührer et al. [4] empirically assessed 15 public malware blacklists and 4 blacklists operated by anti-virus vendors. They found that up to 26.5% of the domains were still missed for the majority of malware families. Only a single blacklist is sufficient to protect against malwares that use DGA.

In addition to these techniques, another family of methods emerges when the use of DGA classification is considered. The basic motivation is to identify malicious domains and more importantly find the related malware structure. In general, DGA categorization can be treated as a multiclass task. It can be either retrospective or real-time detection [3]. Retrospective detection divides domains into clusters to obtain statistical properties such as history of suspicious activities. Real-time detection is a much harder problem; it is based on the sole domain name and linguistic features to build the model [2]. Using linguistic properties has a potential drawback because they can be easily bypassed by the malware author, while deriving a new set of features is rather challenging.

Woodbridge et al. [3] overcame this problem by introducing the feature-less Long Short-Term Memory network (LSTM) [5,6]. This algorithm could be deployed in almost any setting. It also demonstrated to provide a 90% detection rate at a 10^{-4} false positive (FP) rate. While effective, LSTM is naturally sensitive to the multiclass imbalance problem. Due to the time and cost required to collect samples, the non-DGA generated class vastly outnumbers other DGA generated classes; it is common that the imbalance ratio is

* Corresponding author.
E-mail addresses: ductq@soict.hust.edu.vn (D. Tran), hieumd@soict.hust.edu.vn (H. Mac), vantv@soict.hust.edu.vn (V. Tong), anhth@soict.hust.edu.vn (H.A. Tran), giangnl@soict.hust.edu.vn (L.G. Nguyen).

https://doi.org/10.1016/j.neucom.2017.11.018
0925-2312/

on the order of 1:1000. LSTM is accuracy-oriented; it tends to bias towards the prevalent class, leaving a number of DGA families to be undetectable.

This work introduces a novel LSTM.MI algorithm that is robust to imbalanced class distribution. The basic idea is to rely on both binary and multiclass classification models, where LSTM is adapted to include cost items into its backpropagation learning mechanism. The cost items take into account identification importance between classes [7]. It is noticeable that dealing with class imbalance has been an active research area over the last decade. Although numerous solutions were developed, the majority of efforts so far have focused on two-class scenario [8]. Zhou and Liu [9] pointed out that handling multiclass classification poses new challenges and is more difficult than handling the binary one. The LSTM.MI algorithm is investigated through experiments on the real-world collected dataset, where the multiclass imbalance prevails. We observe that this algorithm is able to provide an improvement of at least 7% in terms of macro-averaging recall, precision and F1-score with respect to the original LSTM and other state-of-the-art solutions.

The structure of this paper is as follows. Section 2 provides an overview on DGAs, DGA classification methods and LSTM. The LSTM.MI algorithm is discussed in detail in Section 3. Section 4 presents our empirical results. Section 5 is dedicated to conclusion and future works.

2. Background and related works

2.1. Domain Generation Algorithms

Domain Generation Algorithm (DGA) is based on seeds that involve current data and static integer values in creating a pseudo-random string. It also appends a top level domain (TLD) such as .ru, .com, etc. to establish a proper domain name. The hardcoded IP addresses can be blacklisted and blocked. Modern botnets use DGA to build a resilient command and control (C&C) infrastructure [2]. Using dynamic domains makes it difficult for law enforcement agencies to eliminate botnets [10].

DGA malwares may vary in complexity. *Conficker.C* generates 50,000 domain names and attempts to contact 500 [11]. *W32.Virut* uses public-key cryptography to prevent their commands from being mimicked. *Suppobox* concatenates various words using (typical) English dictionary, while *Ramnit* models the real domain distribution [3]. Since the domain names are short-lived, blacklisting and sinkholing would be ineffective. Detecting DGA botnets becomes a challenging task in computer security.

2.2. DGA classification algorithms

DGA classification is essential in identifying the group related to malware generated domains. It provides a deep understanding to recognize bots that share a similar DGA [2]. Perdisci et al. [12] took advantage of bulk predictions on a subset of domains to derive network features such as number of resolved IPs, IP diversity and prefixes. A C4.5 decision tree was also constructed to distinguish between malicious flux services and other networks.

Bilge et al. [13] proposed EXPOSURE using J48 tree and various time-based, DNS answer-based and domain-based properties. The authors demonstrated that EXPOSURE is able to detect all kinds of non-benign domains including dropzones. McGrath and Gupta [14] measured characteristics including whois records, lexical features and IP addresses of phishing and non-phishing URL. Ma et al. [15] utilized length of domain name, host name and other properties to determine bot, which is used for advertising spam. Yadav et al. [16] computed the distribution of alphabetic characters and bi-grams in all domains. Anatonakakis et al. [1] exploited similar features to indicate the various malware families. The C&C detection was based on Hidden Markov Models and active DGA generated domains.

Salomon and Brustoloni [17] observed a high similarity between domain names that belong to the same DGA. Kwon et al. [18] developed a scalable approach called PsyBoG. The algorithm leveraged the power spectral density (PSD) analysis to discover the major frequencies, given by the periodic DNS queries of botnets. It produced a detection rate of 95% and identified 23 unknown and 26 known malware families with 0.1% false positive (FP). Gu et al. [19] recognized all the *Kraken* and 99.8% of *Conficker* bots using a chain of quantity and linguistic evidence, and traffic that is collected from a single network.

In all the above works, DGA classification is achieved retrospectively. Krishnan et al. [20] argued that retrospective detection is not amenable to immediate applications. It requires several hours before the domain clusters meet the minimum threshold to produce a good performance. Schiavoni et al. [2] presented a real-time DGA detection using two classes of linguistic properties. i.e., meaningful character ratio and n-gram normality score. Nonetheless, the linguistic/statistical features are language dependent and can be easily circumvented by malware author.

2.3. Long Short-Term Memory network

Long Short-Term Memory network (LSTM) [5,6] is a deep Recurrent Neural Network (RNN) that is better than the conventional RNN on tasks involving long time lags [21]. It has been applied in a range of applications such as language modeling [22], speech recognition [23] and DGA botnet detection [3]. LSTM basic unit is the memory block containing one or more memory cells and three multiplicative gating units (see Fig. 1(a)). For the gate, f is a logistic sigmoid, ranging from 0 to 1. Let us assume that w_{lm} is the weight on connection between the source unit m and target unit l with $l \in \{out,\ in,\ \varphi, c_j^v\}$ denoting the output, input, forget gate and the cell, respectively. We also consider discrete time steps t. A single step involves the update of all units (i.e., forward pass) and the error computation of w_{lm} (i.e., backward pass). The cell state $s_{c_j^v}$ is updated based on its current state $z_{c_j^v}$ and other sources of input, i.e., z_φ, z_{in} and z_{out}. The forget gate (φ) is designed to learn to reset blocks once their contents are useless [6].

LSTM aims at minimizing the cost function (we assume softmax) of the network with respect to the actual output y^k of the k-output neuron and the target t^k:

$$E(t) = -\sum_{p \in Samples} \sum_k t^k(t) \log y^k(t) \tag{1}$$

This can be done by using the gradient descent with truncated version of real-time recurrent learning (RTRL) [21]. The weight changes for the output gate, input gate, forget gate and the cell are determined using the learning rate α as follows:

$$\Delta w_{out_j m}(t) \overset{tr}{=} \alpha\, f'_{out_j}\left(z_{out_j}(t)\right)\left(\sum_v s_{c_j^v}(t) \sum_k w_{kc_j^v} \delta_k(t)\right) y^m(t-1) \tag{2}$$

$$\Delta w_{in_j m}(t) = \alpha \sum_v e_{s_{c_j^v}}(t) \frac{\partial s_{c_j^v}(t)}{\partial w_{in_j m}} \tag{3}$$

$$\Delta w_{\varphi_j m}(t) = \alpha \sum_v e_{s_{c_j^v}}(t) \frac{\partial s_{c_j^v}(t)}{\partial w_{\varphi_j m}} \tag{4}$$

$$\Delta w_{c_j^v m}(t) = \alpha\, e_{s_{c_j^v}}(t) \frac{\partial s_{c_j^v}(t)}{\partial w_{c_j^v m}} \tag{5}$$

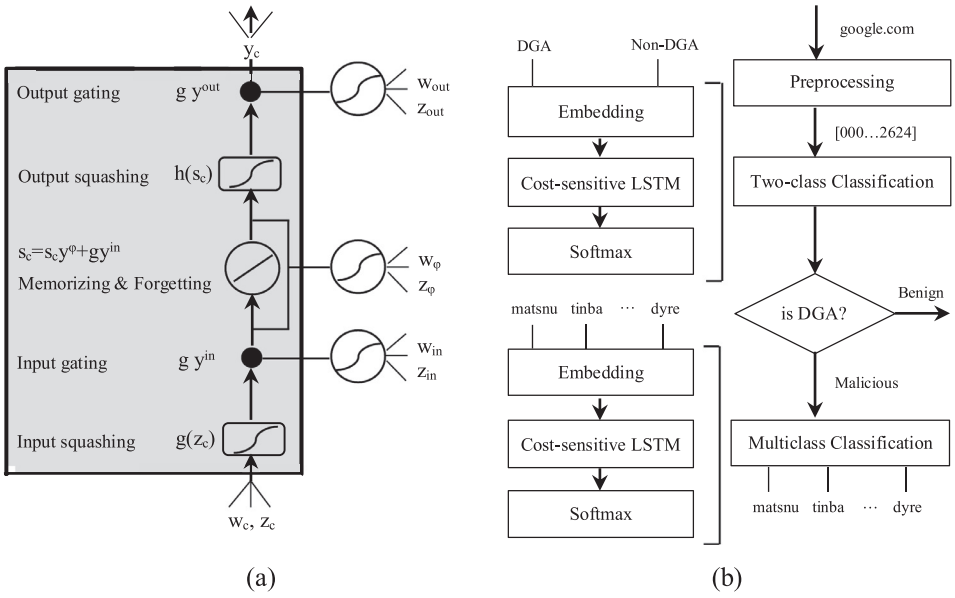

Fig. 1. (a) LSTM memory block with only one cell; (b) the LSTM.MI algorithm.

where

$$\delta_k(t) = -f_k'(z_k(t))\frac{\partial E(t)}{\partial y^k(t)} = f_k'(z_k(t))\frac{t^k(t)}{y^k(t)} \tag{6}$$

and $e_{s_{c_j^v}}$ is the internal state error, which is calculated as

$$e_{s_{c_j^v}}(t) \stackrel{tr}{=} y_{out_j}(t)\left(\sum_k w_{kc_j^v}\delta_k(t)\right) = y_{out_j}(t)\left(\sum_k w_{kc_j^v} f'(z_k(t))\frac{t^k(t)}{y^k(t)}\right) \tag{7}$$

In DGA botnet detection, LSTM accepts variable-length character sequences such that it does not require linguistic/statistical features to be extracted [3]. This algorithm is also compact; the update complexity per weight and time step and storage complexity per weight are on the order of $O(1)$ [6].

3. Learning method

3.1. Proposed cost-sensitive LSTM

The original LSTM treats all the samples equally (see Eq. (1)). Hence, it is naturally quite sensitive to the class imbalance problem. This algorithm is biased towards the prevalent classes, limiting its ability to detect DGA families with very small representation in the training data [3]. Cost-sensitive learning plays a vital role in real-world data mining applications and provides a mean to handle the class imbalance problem [24–29]. Much of the previous research is devoted to make decision trees cost-sensitive [30–32]. Inspired by the success of cost-sensitive C4.5 [30], Jiang et al. [33] introduced the instance-weighting method to induce cost-sensitive Bayesian network classifiers that seek to minimize the overall misclassification costs. Wu and Chang [34] adjusted the SVM decision plane by using different penalty factors for small and prevalent classes, reflecting their importance during training. Zong et al. [35] proposed a weighted Extreme Learning Machine (ELM) for generalized single hidden layer feedforward networks to tackle the multiclass imbalanced class distribution. However, no attempt has been made to discuss cost-sensitive LSTM.

In the literature, it is possible to design a "wrapper" that converts the cost-insensitive LSTM to the cost-sensitive one [9,24]. The most common approach to achieve this objective is to rebalance the training data using either oversampling or undersampling [36]. Oversampling replicates samples in the small classes. It is time-consuming and may lead to overfitting in the model building [37]. Undersampling requires shorter training time. It may risk losing critical information, pertaining the prevalent classes [7,38]. Zhou and Liu [9] observed that resampling is not helpful in dealing with the class imbalance on multiclass tasks.

Threshold-moving manipulates the LSTM outputs by giving the small classes relatively higher impacts on the learning. The decision hyperplane is constructed using the original training data, while cost items are only introduced in the testing phase [9]. On the other hand, AdaBoost.M1 [39] is a method, which is capable of improving the performance of any classifier, provided that the classifier is better than random guessing [7]. This algorithm is accuracy-oriented and its combination with other techniques has led to several proposals to handle the multiclass imbalance [40–42]. Particularly, RUSBoost [41] is based on the idea, where undersampling is integrated into AdaBoost.M1. In [40], RUSBoost was proved to be better, faster and less complex alternative to AdaC2, SMOTEBoost, MSMOTEBoost, UnderBagging, EasyEnsemble, BalanceCascade, and hence, becoming the viable choice to learn from the skewed training data [4,41,43].

Kukar and Kononenko [44] introduced cost-sensitivity to neural network by adapting the error minimization function in order to account for expected costs. Although the authors did not target the class imbalance problem, their method not only maintained the original network structure, but also strengthened the learning with regard to the more expensive classes. Motivated by such pioneering work, the present paper develops an algorithmic approach to directly include the misclassification costs into the LSTM's backward pass. Each sample p is associated with a cost item $C[class(p), k]$, where $class(p)$ and k are the actual and predicted class, respectively. The cost item denotes the classification importance such that lower value is assigned to sample that belongs to the prevalent class [7]. By introducing it into Eq. (1), we have that

$$E(t) = -\sum_{p\in Samples}\sum_k t^k(t)\log y^k(t)\, C[class(p), k] \tag{8}$$

From the backpropagation, it can be seen that such cost item acts as a constant in the partial derivative of $E(t)$. Eqs. (6) and

(7) can be rewritten as

$$\delta_k(t) = -f'_k(z_k(t))\frac{\partial E(t)}{\partial y^k(t)} = f'_k(z_k(t))\frac{t^k(t)}{y^k(t)}C[class(p), k] \quad (9)$$

$$e_{s_{c_j^v}}(t) \overset{tr}{=} y_{out_j}(t)\left(\sum_k w_{kc_j^v} f'_k(z_k(t))\frac{t^k(t)}{y^k(t)}C[class(p), k]\right) \quad (10)$$

The change in Eqs. (6) and (7) results in a change in Eqs. (2)–(5), where $C[class(p), k]$ is taken into account. The cost item controls the magnitude of the weight updates [37]. Samples with larger training error are emphasized, making the learning intentionally biased towards the small classes. The cost-sensitive LSTM is able to address the multiclass imbalance without requiring class decomposition.

Given a data set, the cost matrix is usually unknown. The genetic algorithm can be applied to choose the optimal cost matrix. This is however very time-consuming and may not possible to achieve in practice [8]. For the sake of convenience, we assume that samples in one category are equally costly. $C[i, i]$ denotes the misclassification cost of the class i, which is generated using the class distribution as

$$C[i, i] = \left(\frac{1}{n_i}\right)^{\gamma} \quad (11)$$

where $\gamma \in [0, 1]$ is a trade-off parameter. $\gamma = 1$ implies that $C[i, i]$ is inversely proportional to the class size n_i. The quantity of the small and prevalent classes are rebalanced into the 1:1:...:1 ratio [35]. On the other hand, $\gamma = 0$ implies that the cost-sensitive LSTM reduces to the original LSTM, which is cost-insensitive.

3.2. The essence of cost-sensitive LSTM

The gap between the cost-insensitive and cost-sensitive LSTM can be also interpreted from the movement of the decision boundaries [35]. Consider the multiclass problem with the imbalance ratio 1:8 as an example. The training data consists of 40 *non-DGA* (red circle), 20 *Pykspa* (blue cross) and 5 *Corebot* (black square) samples, which are projected into 2-dimensional space using t-SNE [45]. The non-DGA samples are collected from the Alexa top 100,000 sites.

Fig. 2(a) illustrates the cost-insensitive LSTM that provides an accuracy of 92.5% for the prevalent class, while the accuracies for the small classes are 80% and 0%, respectively. In Fig. 2(b), higher cost items are associated with the *Pykspa* and *Corebot* classes. From the optimization viewpoint, the decision boundaries are pushed towards the prevalent class [35]. A possible drawback is that the cost-sensitive LSTM increases the accuracy (60%) on *Corebot* class, while decreasing the detection rate (87.5%) related to the non-DGA generated domains.

From Fig. 2(c), the larger γ value leads to higher recalls and lower precisions on the small classes. In general, one has to select an appropriate γ such that both these measures are balanced to achieve better overall F1-score and geometric G-mean [7]. The boundary movement phenomena can be explained as either replicating minority samples (oversampling) or removing redundant majority samples (undersampling). It demonstrates the relationship between algorithmic and sampling approaches to cost-sensitive learning. Similar observations can be found in [35].

3.3. Proposed LSTM.MI algorithm

The cost-sensitive LSTM has an intrinsic weakness. It tends to reduce the accuracy on the prevalent non-DGA (*Alexa*) class (see Fig. 2(b) and (c)). This, in turn, interrupts the legitimate Internet traffic since highly secure applications potentially block DNS queries that are assumed to be malicious.

To avoid the above problem, the LSTM.MI algorithm (see Fig. 1(b)) relies on two-class model, where all the abnormal domains are grouped into a single DGA class. It receives in input a domain name d and classifies whether d appears to be automatically generated. The intuitive motivation is to produce the optimal data construction for lowering the imbalance ratio and achieving the high detection rate on the non-DGA class. Once d is deemed to be automatically generated, the algorithm subsequently attempts to assign correct label to the suspicious domain. The fingerprinting is built upon multiclass model; each category is analogous to a malware family. In both the stages, the cost-sensitive LSTM is applied to deal with raw domain names. Fig. 2(d) provides an overview of the effect of LSTM.MI, where γ is set equal to 0.3. In two-class model, the DGA class contains both the *Pykspa* and *Corebot* samples. The imbalance ratio is reduced from 1:8 to 1:1.6. It becomes obvious that the LSTM.MI algorithm is able to produce the accuracies of 85% and 60% for the *Pykspa* and *Corebot* classes, while retaining the high performance (92.5%) on the *Alexa* class.

For simplicity, we generate a dictionary containing all the possible valid characters (underscore, dash, period and alphanumeric). The characters are indexed and the domain d is mapped into a real-valued vector using these indices. The vector is also padded with 0 s to have the length of L. It is then projected to create a matrix $\Re^{DxL}$ in the embedding layer. L denotes the maximum domain length in the training data, while D is a parameter that represents the degree of freedom in the model [3]. In this paper, we choose $D = 128$. It should be noted that increasing D does not necessarily produce higher overall accuracy, but introduces additional computational complexity.

The cost-sensitive LSTM layer aims to extract various features that characterize a given domain name. These features are not hand-crafted, making it hard for adversary to circumvent. Forward pass is the same as in the original LSTM [21]. The backward pass is detailed in Fig. 3. The difference lies on $\delta_k(t)$ and internal state error, which are derived from Eqs. (9) and (10), while the weight updates are computed using Eqs. (2)–(5). To label the given domain name, the softmax layer is applied to minimize the cross-entropy or maximize the log-likelihood [3]. We have that

$$p_i = \frac{\exp(z_{out_i})}{\sum_j \exp(z_{out_j})} \quad (12)$$

The predicted class k would be

$$k = \arg\max_i p_i \quad (13)$$

All the LSTM.MI code is written in Python using Keras library [46] and scikit-learn tools [47].

4. Experiments

4.1. Dataset specification

In the experiments, we evaluate our candidate method on an open dataset containing 1 non-DGA (*Alexa*) and 37 DGA classes. The dataset is collected from two sources: the Alexa top 100,000 sites with/without the www. child label [3,48] and the OSINT DGA feed from Bambenek Consulting [49].

In Table 1, the dataset exhibits different imbalance degrees, ranging from 1:2 to 1:3534. It also involves some notable DGA families such as *Cryptolocker, Locky, Kraken, Gameover Zeus. Matsnu, Cryptowall, Suppobox* and *Volatile* malwares are based on pronounceable domains, which are randomly generated using (typical) English dictionary with a narrow randomization space. The five-fold cross validation (CV) is carried out since several classes are very small. This is to ensure that each fold has at least five samples from every class.

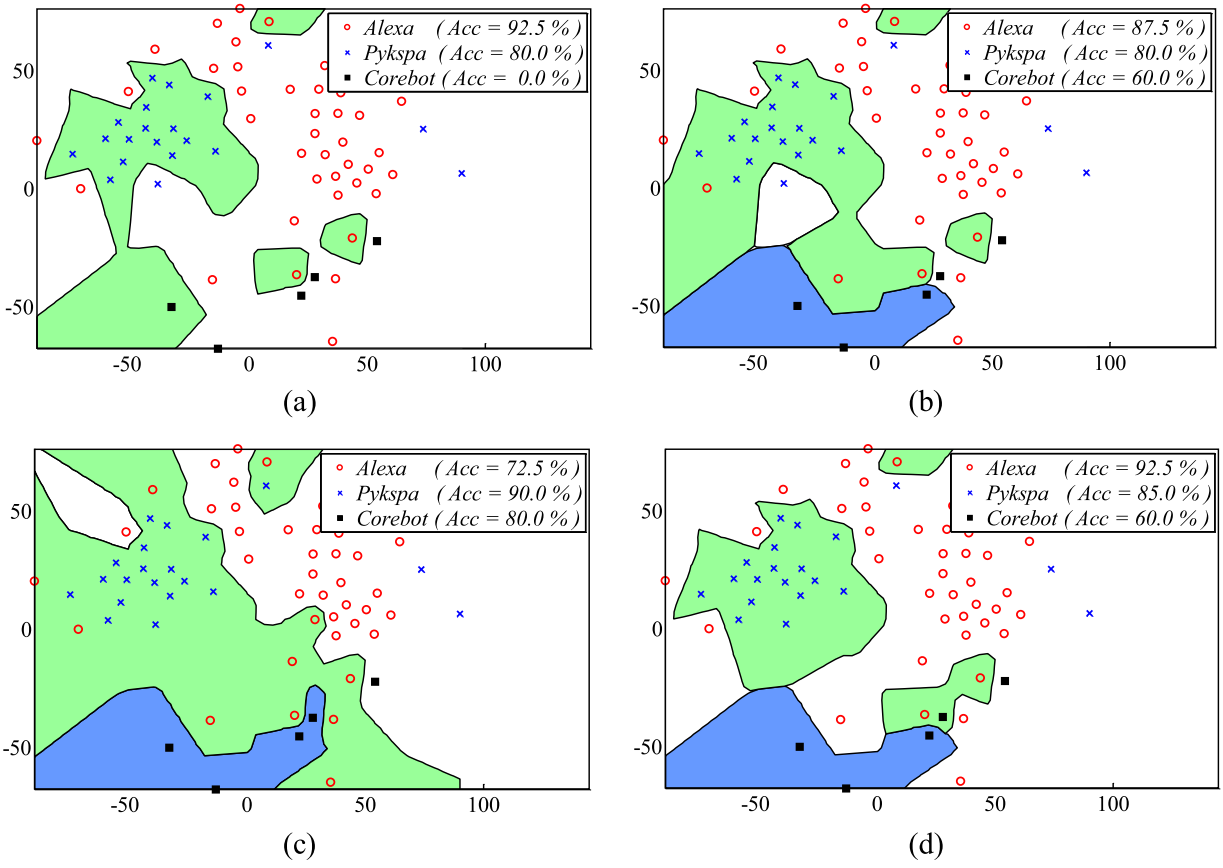

Fig. 2. The decision boundaries produced by the (a) cost-insensitive LSTM, (b) cost-sensitive LSTM ($\gamma = 0.3$), (c) cost-sensitive LSTM ($\gamma = 1$) and (d) LSTM.MI ($\gamma = 0.3$). The imbalance ratio is 1:8.

Table 1
Summary of the collected dataset (Source: the Alexa top 100,000 sites [48] and the OSINT DGA feed from Bambenek Consulting [49]).

Domain type	# Sample	Pronounceable	Domain type	# Sample	Pronounceable
Geodo	58	✗	*Fobber*	60	✗
Beebone	42	√	*Alexa*	88,347	√
Murofet	816	✗	*Dyre*	800	✗
Pykspa	1422	✗	*Cryptowall*	94	√
Padcrypt	42	✗	*Corebot*	28	✗
Ramnit	9158	✗	*P*	200	✗
Volatile	50	√	*Bedep*	172	✗
Ranbyus	1232	✗	*Matsnu*	48	√
Qakbot	4000	✗	*PT Goz*	6600	✗
Simda	1365	✗	*Necurs*	2398	✗
Ramdo	200	✗	*Pushdo*	168	✗
Suppobox	101	√	*Cryptolocker*	600	✗
Locky	186	✗	*Dircrypt*	57	✗
Tempedreve	25	✗	*Shifu*	234	✗
Qadars	40	✗	*Bamital*	60	✗
Symmi	64	✗	*Kraken*	508	✗
Banjori	42,166	✗	*Nymaim*	600	✗
Tinba	6385	✗	*Shiotob*	1253	✗
Hesperbot	192	✗	*W32.Virut*	60	✗

4.2. Performance measures

In general, a classifier can be gauged using the overall accuracy. Unfortunately, in multiclass imbalanced data, this metric is no longer a proper measure since the prevalent classes have much higher impacts with respect to the small classes [7,37,41]. To give more insight into the accuracy obtained within a given class, precision, recall, F1-score are adopted in this paper. Precision is the percentage of relevant domains that are retrieved, while recall is the percentage of retrieved domains that are relevant. F1-score represents a harmonic mean between precision and recall.

As illustrated in Table 2, quality of the overall classification can be analyzed in two ways. In macro-averaging, a metric is averaged over all classes that are treated equally. Micro-averaging is based on the cumulative True Positive, False Positive, True Negative and False Negative [50]. It favors the classes that have more samples (prevalent classes). Macro-averaging seems to be better indicator in multiclass imbalance problem. However, micro-averaging

Algorithm 1 Cost-sensitive LSTM: Backward pass

errors (Eq. 8) and δs
  D represents the number of memory blocks
  S_j represents the number of cells in block j
δs of output units (Eq. 9): $\delta_k = f'_k(z_k)\frac{t^k}{y^k}C[class(p),k]$
for $j=1$ to D step 1 **do**
  δs of output gates: $\delta_{out_j} = f'_{out_j}(z_{out_j})\left(\sum_{\upsilon} s_{c_j^\upsilon}\sum_k w_{kc_j^\upsilon}\delta_k\right)$
  for $\upsilon=1$ to S_j step 1 **do**
    Internal state error (Eq. 10): $e_{s_{c_j^\upsilon}}^{tr} = y_{out_j}\left(\sum_k w_{kc_j^\upsilon} f'_k(z_k)\frac{t^k}{y^k}C[class(p),k]\right)$
  end for
end for
The standard backpropagation ***weight changes***: $\Delta w_{km} = \alpha\delta_k y^m$
for $j=1$ to D step 1 **do**
  Output gates (Eq. 2): $\Delta w_{out_j m} = \alpha\delta_{out_j} y^m$
  Input gates (Eq. 3): $\Delta w_{in_j m} = \alpha\sum_{\upsilon} e_{s_{c_j^\upsilon}} dS_{in_j m}^{j\upsilon}$
  Forget gates (Eq. 4): $\Delta w_{\varphi_j m} = \alpha\sum_{\upsilon} e_{s_{c_j^\upsilon}} dS_{\varphi_j m}^{j\upsilon}$
  for $\upsilon=1$ to S_j step 1 **do**
    Cells (Eq. 5): $\Delta w_{c_j^\upsilon m} = \alpha\, e_{s_{c_j^\upsilon}} dS_{c_j^\upsilon m}^{j\upsilon}$
  end for
end for

Fig. 3. The cost-sensitive LSTM's backward pass in Pseudo-code.

Table 2
Micro-averaging and macro-averging precision, recall and F1-score [50]. *i* denotes the class index, while μ and *M* denote micro- and macro-averaging, respectively.

Micro-averaging	Macro-averaging
$\text{Precision}_\mu = \frac{\sum_i True\ Positive_i}{\sum_i (True\ Positive_i + False\ Positive_i)}$	$\text{Precision}_M = \frac{\sum_i \frac{True\ Positive_i}{True\ Positive_i + False\ Positive_i}}{\text{Number of classes}}$
$\text{Recall}_\mu = \frac{\sum_i True\ Positive_i}{\sum_i (True\ Positive_i + False\ Negative_i)}$	$\text{Recall}_M = \frac{\sum_i \frac{True\ Positive_i}{True\ Positive_i + False\ Negative_i}}{\text{Number of classes}}$
$\text{F1}-\text{score}_\mu = \frac{2}{1/\text{Precision}_\mu + 1/\text{Recall}_\mu}$	$\text{F1}-\text{score}_M = \frac{2}{1/\text{Precision}_M + 1/\text{Recall}_M}$

would be included in this paper for interested readers. It can be noted that micro-averaging recall is equal to the overall accuracy, reported in the literature.

4.3. Cost-sensitive LSTM

Finding the appropriate cost matrix to produce the best classification performance in accordance to the learning objective is a difficult task. This paper assumes that every class has its own misclassification cost. The cost items are used to boost the detection rate on the small classes. Since the optimal γ is not available, various $\gamma \in [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]$ values should be tested. Note that at $\gamma = 0$ the cost-sensitive LSTM reduces to the cost-insensitive one.

In two-class task, the macro-averaging is mostly identical to the micro-averaging. Thus, Fig. 4 only illustrates the macro-averaging precision, recall and F1-score. As it can be seen, the difference between these measures is relatively small. There is almost no change in the classification performance as the value of γ is increased. This is due the fact that malicious domains are grouped into a single class. The imbalance ratio (IR) reduces to 1:1.08. A change in γ does not induce a significant change in the cost items associated with the various classes.

In multiclass task, the *Alexa* samples are completely removed from the training dataset. Each class is equivalent to a malware family. From Fig. 5(a) and (b), we observe that the highest macro-averaging precision is achieved at $\gamma = 0.1$. As γ grows larger, the decision boundary is pushed towards the prevalent classes. This increases the recall, and at the same time decreases the precision on the small classes. Obviously, there is a trade-off between the recall and precision values. An undesirable fact is that a certain amount of majority samples are misclassified, leading to a reduction in the overall accuracy. As it is expected, $\gamma = 1$ produces the lowest micro-averaging recall (0.7819) and F1-score (0.7830). In Fig. 5(c), 9 DGA families cannot be detected at $\gamma = 0$. This number is 2 at $\gamma = 1$. This implies that the cost-sensitive LSTM augments the chance to recognize additional DGA families. It is shown to be better alternative to the cost-insensitive counterpart that only attains 0.496 macro-averaging F1-score. Based on the above observations, we select $\gamma = 0.3$ to provide a balanced solution between micro- and macro-averaging. The highest macro-averaging

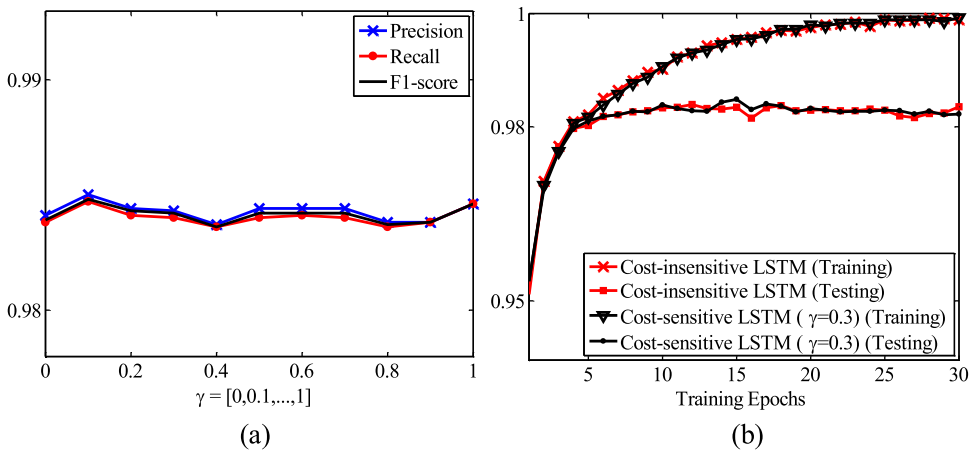

Fig. 4. Performance of the cost-sensitive LSTM on two-class task. (a) Macro-averaging precision, recall and F1-score with respect to the $\gamma \in [0, 1]$, and (b) learning curves related to the cost-insensitive LSTM and cost-sensitive LSTM ($\gamma = 0.3$).

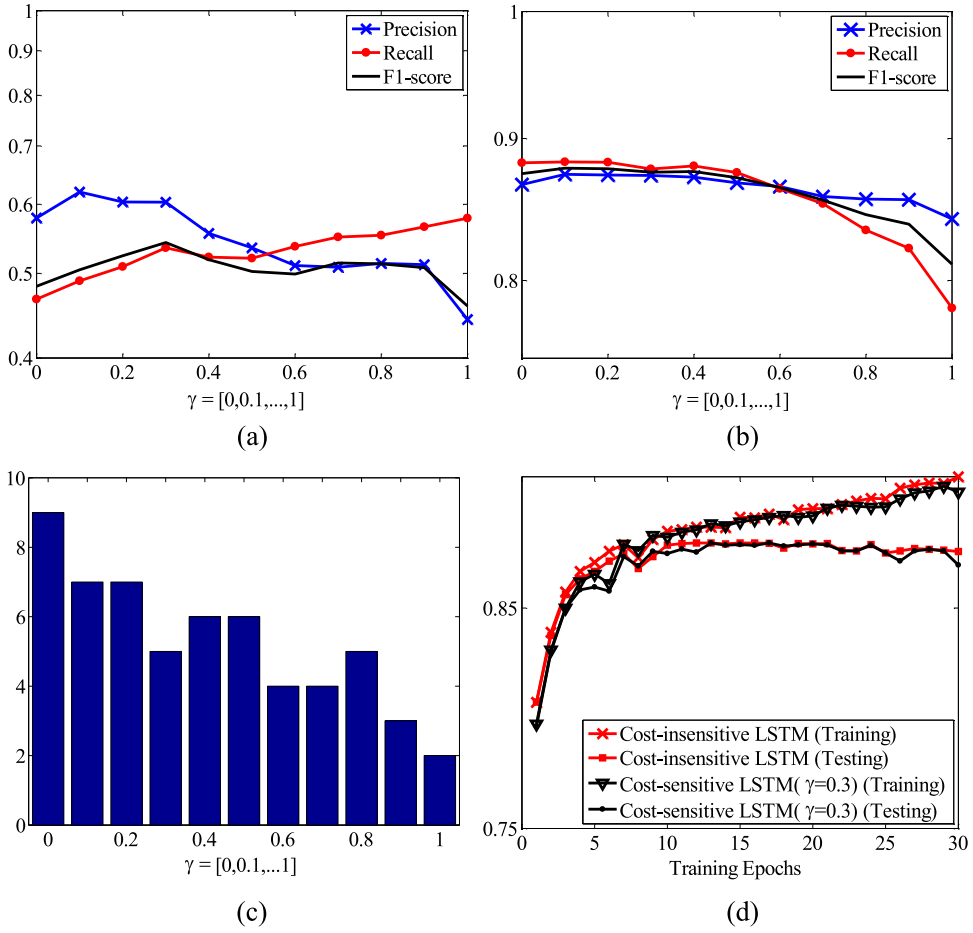

Fig. 5. Performance of the cost-sensitive LSTM on multiclass task. (a) Macro-averaging precision, recall and F1-score, (b) micro-averaging precision, recall and F1-score, (c) the number of DGA families that cannot be recognized with respect to the $\gamma \in [0, 1]$, and (d) learning curves relating to the cost-insensitive LSTM and cost-sensitive LSTM ($\gamma = 0.3$).

accuracy (0.6034 precision, 0.535 recall and 0.5671 F1-score) for multiclass task is also achieved at $\gamma = 0.3$.

Convergence and generalization are important properties of the cost-sensitive LSTM. Convergence is defined as the number of epochs, required to achieve a sufficiently good solution during training. Generalization measures the detection capability on new unseen data [51]. In all experiments, the learning rate α is kept the same in order to observe the effect of varying cost items ($\gamma \in \{0, 0.3\}$).

From Figs. 4(b) and 5(d), both the binary and multiclass cost-sensitive LSTM appear to have converged after 17 epochs. The training time is remarkably fast, requiring just an hour on an Intel core i5 8GB RAM computer. We observe that the cost-sensitive LSTM is slightly prone to overfitting. After 18

Table 3
Performance comparison of the cost-sensitive learning methods on two-class task. The number in boldface denotes the highest F1-score.

	Oversampling			Threshold-moving			RUSBoost			LSTM ($\gamma = 0.3$)		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Non-DGA	0.9853	0.9773	0.9813	0.9849	0.9761	0.9805	0.9836	0.9810	0.9823	0.9816	0.9881	**0.9849**
DGA	0.9756	0.9842	0.9799	0.9744	0.9837	0.9790	0.9794	0.9823	0.9809	0.9870	0.9799	**0.9845**
Micro-averaging	0.9806	0.9806	0.9806	0.9799	0.9797	0.9798	0.9816	0.9816	0.9816	0.9842	0.9842	**0.9842**
Macro-averaging	0.9805	0.9808	0.9806	0.9797	0.9799	0.9798	0.9815	0.9817	0.9816	0.9843	0.9840	**0.9842**

Table 4
Performance comparison of the cost-sensitive learning methods on multiclass task. The number in boldface denotes the highest F1-score.

	Oversampling			Threshold-moving			RUSBoost			LSTM ($\gamma = 0.3$)		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Geodo	0.1429	0.0833	**0.1053**	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Beebone	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Murofet	0.4291	0.6503	0.5171	0.7524	0.4847	0.5896	0.5355	0.5092	0.5220	0.5330	0.7423	**0.6205**
Pykspa	0.7374	0.7218	0.7295	0.7985	0.7535	**0.7754**	0.6889	0.6549	0.6715	0.8023	0.7430	0.7715
Padcrypt	0.9091	0.8333	**0.8696**	0.6875	0.9167	0.7857	0.8000	0.6667	0.7273	1.0000	0.7500	0.8571
Ramnit	0.5560	0.3903	0.4586	0.5733	0.8859	**0.6961**	0.4350	0.3619	0.3951	0.6068	0.8062	0.6925
Volatile	0.9091	1.0000	0.9524	0.8333	1.0000	0.9091	0.8333	1.0000	0.9091	1.0000	1.0000	**1.0000**
Ranbyus	0.3262	0.4309	0.3713	0.4029	0.3374	0.3673	0.2396	0.3293	0.2774	0.3617	0.7073	**0.4787**
Qakbot	0.5240	0.4637	0.4920	0.7245	0.5062	**0.5960**	0.3795	0.4625	0.4169	0.7716	0.4350	0.5564
Simda	0.9536	0.9780	0.9656	0.9681	1.0000	0.9838	0.9574	0.9890	0.9730	0.9579	1.0000	**0.9785**
Ramdo	0.8864	0.9750	0.9286	1.0000	1.0000	**1.0000**	0.9091	1.0000	0.9524	0.9524	1.0000	0.9756
Suppobox	0.4706	0.4000	0.4324	0.3500	0.3500	0.3500	0.4091	0.4500	0.4286	0.4167	0.5000	**0.4545**
Locky	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0172	0.0270	0.0211	0.0000	0.0000	0.0000
Tempedreve	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Qadars	0.8333	0.6250	0.7143	1.0000	0.5000	0.6667	0.5000	0.6250	0.5556	1.0000	0.6250	**0.7692**
Symmi	0.4000	0.3077	**0.3478**	0.0000	0.0000	0.0000	0.0870	0.1538	0.1111	0.5000	0.1538	0.2353
Banjori	0.9986	0.9992	**0.9989**	0.9992	1.0000	0.9996	0.9981	0.9740	0.9859	0.9988	1.0000	**0.9994**
Tinba	0.8319	0.8794	0.8550	0.8947	0.9984	**0.9437**	0.8386	0.5450	0.6607	0.8951	0.9961	0.9429
Hesperbot	0.0938	0.0789	**0.0857**	0.0000	0.0000	0.0000	0.0270	0.0263	0.0267	0.3333	0.0263	0.0488
Fobber	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Dyre	0.9810	0.9688	0.9748	0.9639	1.0000	0.9816	1.0000	0.9938	**0.9969**	0.9816	1.0000	0.9907
Cryptowall	0.2667	0.2105	0.2353	0.5000	0.1053	0.1739	0.0714	0.1579	0.0984	0.6250	0.2632	**0.3704**
Corebot	0.5000	0.5000	0.5000	0.6667	0.3333	0.4444	0.4000	0.6667	0.5000	0.7500	0.6000	**0.6667**
P	0.3600	0.2250	0.2769	0.4878	0.5000	**0.4938**	0.3115	0.4750	0.3762	0.3333	0.5500	0.4151
Bedep	0.4737	0.5294	0.5000	0.7083	0.5000	**0.5862**	0.4412	0.4412	0.4412	0.6875	0.3235	0.4400
Matsnu	0.8333	0.5000	0.6250	0.5714	0.4000	0.4706	0.2353	0.4000	0.2963	1.0000	0.7000	**0.8235**
PT Goz	0.9970	0.9977	0.9973	0.9962	1.0000	0.9981	0.9977	0.9909	0.9943	0.9985	1.0000	**0.9992**
Necurs	0.1318	0.2463	0.1718	0.5000	0.0960	0.1611	0.1246	0.1837	0.1485	0.5248	0.1104	**0.1824**
Pushdo	0.5000	0.5000	0.5000	0.5714	0.5882	0.5797	0.4694	0.6765	0.5542	0.6571	0.6765	**0.6667**
Cryptolocker	0.1295	0.1500	**0.1390**	0.0000	0.0000	0.0000	0.0658	0.1667	0.0943	0.2000	0.0167	0.0308
Dircrypt	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Shifu	0.3167	0.4043	0.3551	0.4111	0.7872	**0.5401**	0.3220	0.4043	0.3585	0.3711	0.7660	0.5000
Bamital	0.6923	0.7500	0.7200	1.0000	0.6667	0.8000	0.9231	1.0000	**0.9600**	1.0000	0.7500	0.8571
Kraken	0.0431	0.0490	0.0459	0.0000	0.0000	0.0000	0.0718	0.1275	**0.0919**	0.0800	0.0196	0.0315
Nymaim	0.1417	0.1500	0.1457	0.2727	0.1000	0.1463	0.1445	0.2083	0.1706	0.2989	0.2167	0.2512
Shiotob	0.9469	0.9243	0.9355	0.9672	0.9402	**0.9535**	0.7993	0.9044	0.8486	0.9741	0.9004	0.9358
W32.Virut	0.2727	0.2500	0.2609	0.0000	0.0000	0.0000	0.3333	0.3333	0.3333	0.7143	0.4167	**0.5263**
Micro-averaging	0.8302	0.8194	0.8248	0.8643	0.8803	0.8722	0.8048	0.7724	0.7883	0.8728	0.8775	**0.8751**
Macro-averaging	0.5024	0.4911	0.4967	0.5298	0.4797	0.5035	0.4423	0.4839	0.4622	0.6034	0.5350	**0.5671**

epochs, the performance on the training set continues to rise while that on the testing set starts decreasing. The cost-sensitive and cost-insensitive learning have somewhat similar overall accuracy. It should be noted that the original LSTM is accuracy-oriented with the aim to maximize the detection rate over all classes.

As already discussed, it is able to design a "wrapper" to make LSTM cost-sensitive. The wrapper methods can be oversampling, Threshold-moving and RUSBoost. Undersampling is already integrated into RUSBoost, and hence, it is omitted in this work. Threshold-moving is based on type (c) cost matrix, where at least one $C[i, j] = 1$ and the unit cost is the minimum cost. We refer the readers to [9,30] for more details. In two-class task (see Table 3), there is no notable difference between the algorithms since the imbalance problem (1:1.08) is not significant. The cost-sensitive LSTM achieves the highest accuracy. This is followed by RUSBoost. Such meta-technique combines 50 weak learners (LSTMs) to improve the generalization capabilities. It however leads to an increased computational complexity.

In multiclass task (see Table 4), Threshold-moving cannot recognize 10 DGA families. Its performance is mostly identical to the accuracy achieved by the cost-insensitive LSTM (i.e., $\gamma = 0$). RUSBoost demonstrates to be the worst performer. The rationale for this is that RUSBoost randomly removes samples to balance the highly skewed class distribution. It may risk losing important information that characterizes the prevalent classes. Overall, $\gamma = 0.3$ leads to the highest overall macro-averaging recall, precision and F1-score. Hence, it is reasonable to directly introduce the cost items into the backpropagation learning mechanism.

4.4. The LSTM.MI algorithm

In LSTM.MI, both binary and multiclass cost-sensitive LSTM work together in combination. Its basic motivation is to retain the high accuracy on the prevalent non-DGA class, while increasing the macro-averaging F1-score on other DGA classes. This section is dedicated to investigate LSTM.MI and highlight its advantages

Table 5
Precision, recall and F1-score for CS-NN, CS-SVM, CS-C4.5 and WELM from five-fold cross validation.

	CS-NN			CS-SVM			CS-C4.5			WELM		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Geodo	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0303	0.0833	0.0444
Beebone	0.0000	0.0000	0.0000	0.6667	1.0000	0.8000	0.8000	1.0000	0.8889	0.8000	0.5000	0.6154
Murofet	0.8000	0.0491	0.0925	0.5385	0.1718	0.2605	0.5814	0.6135	0.5970	0.3372	0.3558	0.3463
Pykspa	0.0000	0.0000	0.0000	0.3231	0.0739	0.1203	0.5833	0.4930	0.5344	0.3696	0.3592	0.3643
Padcrypt	0.0000	0.0000	0.0000	0.2500	0.1667	0.2000	0.3333	0.4167	0.3704	0.1667	0.0833	0.1111
Ramnit	0.4645	0.7631	0.5774	0.4931	0.7238	0.5866	0.4642	0.4771	0.4705	0.4510	0.4694	0.4600
Volatile	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.8182	0.9000	0.8571	0.0000	0.0000	0.0000
Ranbyus	0.0000	0.0000	0.0000	0.3501	0.6789	0.4620	0.3801	0.4512	0.4126	0.4377	0.4715	0.4540
Qakbot	0.7164	0.3063	0.4291	0.6520	0.2600	0.3718	0.4141	0.3675	0.3894	0.3796	0.3450	0.3615
Simda	0.3049	0.0916	0.1408	0.5287	0.3370	0.4116	0.3119	0.3370	0.3239	0.3020	0.2234	0.2568
Ramdo	0.0000	0.0000	0.0000	0.3077	0.2000	0.2424	0.5385	0.5250	0.5316	0.3750	0.3000	0.3333
Suppobox	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Locky	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0256	0.0270	0.0263
Tempedreve	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Qadars	0.0000	0.0000	0.0000	0.1667	0.1250	0.1429	0.3125	0.6250	0.4167	0.2500	0.2500	0.2500
Symmi	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Banjori	0.9306	0.9107	0.9205	0.9641	0.9808	0.9724	0.9995	0.9995	0.9995	0.9976	0.9999	0.9988
Tinba	0.7525	0.9311	0.8323	0.7900	0.9749	0.8728	0.8063	0.8575	0.8311	0.7978	0.8434	0.8199
Hesperbot	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0250	0.0263	0.0256
Fobber	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Alexa	0.9170	0.9675	0.9416	0.9515	0.9730	0.9621	0.9589	0.9650	0.9619	0.9564	0.9606	0.9585
Dyre	0.7571	0.9938	0.8595	0.9933	0.9250	0.9579	1.0000	1.0000	1.0000	0.9815	0.9938	0.9876
Cryptowall	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Corebot	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
P	1.0000	0.2500	0.4000	0.3333	0.1000	0.1538	0.5517	0.4000	0.4638	0.3514	0.3250	0.3377
Bedep	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0938	0.0882	0.0909
Matsnu	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
PT Goz	0.9613	0.9970	0.9788	0.9831	0.9682	0.9756	0.9894	0.9917	0.9905	0.9751	0.9803	0.9777
Necurs	0.0833	0.0021	0.0041	0.2909	0.0333	0.0598	0.1381	0.1396	0.1389	0.1317	0.1125	0.1213
Pushdo	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0789	0.0882	0.0833	0.0303	0.0294	0.0299
Cryptolocker	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0755	0.0667	0.0708	0.0672	0.0667	0.0669
Dircrypt	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Shifu	0.1765	0.0638	0.0937	0.1776	0.4043	0.2468	0.1739	0.1702	0.1720	0.2090	0.2979	0.2456
Bamital	0.0000	0.0000	0.0000	1.0000	0.6667	0.8000	1.0000	1.0000	1.0000	0.9091	0.8333	0.8696
Kraken	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0500	0.0294	0.0370	0.0526	0.0588	0.0556
Nymaim	0.0000	0.0000	0.0000	1.0000	0.0083	0.0165	0.0748	0.0667	0.0705	0.0667	0.0417	0.0513
Shiotob	0.3889	0.6693	0.4919	0.5292	0.5418	0.5354	0.7930	0.7171	0.7531	0.6777	0.5697	0.6190
W32.Virut	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Micro-averaging	0.8309	0.8624	0.8463	0.8740	0.8884	0.8811	0.8792	0.8834	0.8811	0.8707	0.8739	0.8723
Macro-averaging	0.2172	0.1841	0.1993	0.3234	0.2714	0.2951	0.3481	0.3605	0.3517	0.2959	0.2814	0.2884

over the cost-sensitive neural network (CS-NN) [44], cost-sensitive support vector machines (CS-SVM) [34], cost-sensitive C4.5 (CS-C4.5) [30] and Weighted Extreme Learning Machine (WELM) [35]. We also compare such algorithm with other state-of-the-art DGA detection methods, namely C5.0 decision tree (C5.0) [52], Hidden Markov Model (HMM) [1] and the original LSTM (LSTM) [3].

HMM is similar to the original LSTM, which can directly operate on the raw domains. We train one distinct HMM per class. The number of hidden states is set equal to the average length of the domain names in the training dataset. CS-NN, CS-SVM CS-C4.5, WELM and C5.0 represent the retrospective group of methods, which construct the classification model using hand-crafted features, i.e., entropy of character distribution, n-gram normality score ($n \in [1, 2, 3, 4, 5]$), length of domain name, and meaningful character ratio [1,12,13]. CS-C4.5 relies on the instance-weighting method to modify the decision tree splitting criterion that makes C4.5 cost-sensitive [33].

In Tables 5 and 6, HMM produces worse accuracy than expected. According to Woodbridge et al. [3], a possible reason may be due to the collected dataset that involves 37 different DGA families. We note that in [1], HMM was only evaluated using *Conficker, Murofet, Bobax*, and *Sinowal*. The dominance of LSTM.MI is established with a large margin (7% macro-averaging F1-score improvement) compared to the original LSTM. LSTM.MI is proved to be most robust against the multiclass imbalance problem. It is also capable of retaining the high precision (0.9816) and recall (0.9881) on the *Alexa* (non-DGA) class. CS-NN, CS-SVM CS-C4.5, WELM and C5.0 demonstrate to be inferior to both the original LSTM and LSTM.MI.

In Tables 7 and 8, the Wilcoxon signed ranks test has been done to compare each pair of algorithms based on their F1-score on the various non-DGA and DGA data classes [53]. The detailed ranks are shown in Table 7. The sum of ranks below the diagonal is denoted as R^+, while that above the diagonal is denoted as R^- [54]. For a confidence level of $\alpha = 0.05$ and $N = 38$ data classes, the two methods are considered to be significantly different if the smaller of R^+ and R^- is equal or less than 235. Table 8 illustrates the summary of the Wilcoxon test. It is obvious that LSTM.MI achieves the highest performance and the difference between LSTM.MI and other algorithms is significant.

Table 9 shows the evaluation time of the various methods. In most cases, the training is carried out off-line; hence, it is not of interest. For practical uses, evaluation may be critical [43]. As illustrated in Table 9, there is almost no computation cost in LSTM (9 ms). LSTM.MI has to render the decision using the outcomes of both the binary and multiclass cost-sensitive models, thus, requiring more evaluation time (18 ms) to process a domain. The evaluation time of WELM is not given because it is executed using Matlab. CS-NN, CS-SVM, CS-C4.5 and C5.0 are computationally expensive. This is due to the fact that these methods require

Table 6
Precision, recall and F1-score for HMM, C5.0 decision tree, LSTM and LSTM.MI from five-fold cross validation.

	HMM			C5.0			LSTM			LSTM.MI ($\gamma = 0.3$)		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Geodo	0.0127	0.4167	0.0246	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Beebone	0.0308	0.7500	0.0591	0.6250	1.0000	0.7692	1.0000	0.9375	0.9667	0.9046	1.0000	0.9499
Murofet	0.8235	0.2577	0.3925	0.3810	0.4706	0.4211	0.5894	0.6013	0.5952	0.6977	0.6563	0.6755
Pykspa	0.3090	0.1937	0.2381	0.0000	0.0000	0.0000	0.8301	0.6708	0.7420	0.8558	0.7087	0.7752
Padcrypt	0.2069	1.0000	0.3429	0.0000	0.0000	0.0000	0.9445	0.5834	0.7143	0.9286	0.9667	0.9443
Ramnit	0.1081	0.0551	0.0730	0.0000	0.0000	0.0000	0.5627	0.8163	0.6661	0.6070	0.8260	0.6998
Volatile	0.0136	0.6000	0.0267	0.0000	0.0000	0.0000	0.8889	0.6500	0.7434	1.0000	0.5540	0.7130
Ranbyus	0.0424	0.2236	0.0713	0.0000	0.0000	0.0000	0.4140	0.4289	0.4180	0.4324	0.6646	0.5239
Qakbot	0.1240	0.0587	0.0797	0.9773	0.9835	0.9804	0.7109	0.5069	0.5918	0.7669	0.5290	0.6260
Simda	0.0137	0.1465	0.0250	0.7685	0.9640	0.8552	0.9079	0.8993	0.9035	0.8907	0.9010	0.8958
Ramdo	0.0388	0.7250	0.0737	0.0000	0.0000	0.0000	0.9518	0.9875	0.9693	0.9759	0.9537	0.9646
Suppobox	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.3542	0.0934	0.1471
Locky	0.0000	0.0000	0.0000	0.3492	0.2767	0.3088	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Tempedreve	0.0015	0.8000	0.0031	0.9507	0.9766	0.9635	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Qadars	0.0309	0.7500	0.0594	1.0000	1.0000	1.0000	0.0000	0.0000	0.0000	0.9000	0.4318	0.5744
Symmi	0.0065	0.1538	0.0125	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.3750	0.0857	0.1389
Banjori	0.9143	0.1051	0.1885	0.6667	0.2857	0.4000	0.9992	0.9999	0.9995	0.9984	1.0000	0.9991
Tinba	0.0000	0.0000	0.0000	0.6000	0.4167	0.4918	0.8846	0.9456	0.9141	0.8921	0.9814	0.9345
Hesperbot	0.0037	0.0526	0.0069	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Fobber	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Alexa	1.0000	0.0002	0.0003	0.9899	0.9868	0.9883	0.9746	0.9924	0.9834	0.9816	0.9881	0.9849
Dyre	0.9697	1.0000	0.9846	0.1646	0.0567	0.0844	0.9611	0.9969	0.9786	0.9746	1.0000	0.9872
Cryptowall	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Corebot	0.0017	0.4000	0.0035	0.3116	0.2191	0.2573	0.0000	0.0000	0.0000	0.8000	0.5000	0.5857
P	0.2727	0.2250	0.2466	0.0645	0.0140	0.0230	0.5718	0.3625	0.4434	0.5055	0.4405	0.4708
Bedep	0.0060	0.1471	0.0115	0.0000	0.0000	0.0000	0.9000	0.2353	0.3723	0.7322	0.3936	0.5117
Matsnu	0.0000	0.0000	0.0000	0.0800	0.0435	0.0563	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
PT Goz	0.9811	0.6682	0.7950	0.9091	1.0000	0.9524	0.9978	0.9989	0.9983	0.9975	0.9986	0.9980
Necurs	0.0244	0.0729	0.0366	0.0000	0.0000	0.0000	0.4001	0.0875	0.1424	0.4565	0.0982	0.1603
Pushdo	0.0036	0.2353	0.0071	0.1071	0.0268	0.0429	0.6762	0.1912	0.2969	0.5852	0.4917	0.5341
Cryptolocker	0.0163	0.6917	0.0318	0.6406	0.5538	0.5940	0.0000	0.0000	0.0000	0.1111	0.0079	0.0147
Dircrypt	0.0017	0.0909	0.0034	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Shifu	0.0250	1.0000	0.0489	0.2222	0.2000	0.2105	0.3671	0.2872	0.3115	0.3294	0.6518	0.4330
Bamital	0.6316	1.0000	0.7742	0.4839	0.5797	0.5275	0.9445	0.5417	0.6751	1.0000	0.5750	0.7143
Kraken	0.0041	0.0196	0.0068	0.4545	0.4545	0.4545	0.0625	0.0049	0.0091	0.1500	0.0126	0.0233
Nymaim	0.0085	0.2250	0.0165	0.3062	0.3900	0.3431	0.3118	0.0625	0.1031	0.1787	0.1080	0.1342
Shiotob	0.2404	0.2749	0.2565	0.4767	0.3761	0.4205	0.9132	0.8785	0.8955	0.9285	0.8774	0.9022
W32.Virut	0.0035	1.0000	0.0070	0.4403	0.2439	0.3139	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Micro-averaging	0.8085	0.0782	0.1426	0.8652	0.8854	0.8751	0.9175	0.9295	0.9235	0.9261	0.9336	0.9298
Macro-averaging	0.1808	0.3510	0.2386	0.3150	0.3031	0.3089	0.4675	0.3860	0.4229	0.5344	0.4604	0.4946

Table 7
Ranks computed by the Wilcoxon test.

Algorithm	CS-NN	CS-SVM	CS-C4.5	WELM	HMM	C5.0	LSTM	LSTM.MI
CS-NN	–	143.0	93.0	120.5	301.0	228.0	52.5	18.0
CS-SVM	560.0	–	167.0	239.5	456.5	335.0	107.0	35.0
WELM	610.0	536.0	–	531.0	629.0	424.0	221.0	122.0
CS-C4.5	620.5	501.5	172.0	–	634.5	378.5	181.0	106.5
HMM	402.0	284.5	74.0	106.5	–	210.5	127.0	71.0
C5.0	475.0	368.0	317.0	362.5	492.5	–	248.5	208.0
LSTM	688.5	596.0	482.0	522.0	576.0	454.5	–	103.0
LSTM.MI	685.0	668.0	581.0	634.5	670.0	495.0	600.0	–

Table 8
Summary of the Wilcoxon test. • implies that the algorithm in the row improves the algorithm of the column, while ◦ implies that the algorithm in the column improves the algorithm of the row. Lower diagonal level of significance $\alpha = 0.95$; Upper diagonal level of significance $\alpha = 0.9$.

Algorithm	CS-NN	CS-SVM	CS-C4.5	WELM	HMM	C5.0	LSTM	LSTM.MI
CS-NN	–	◦	◦	◦		◦	◦	◦
CS-SVM	•	–	◦	◦			◦	◦
WELM	•	•	–	•	•		◦	◦
CS-C4.5	•		◦	–	•		◦	◦
HMM			◦	◦	–	◦	◦	◦
C5.0					•	–		◦
LSTM	•	•	•	•	•		–	◦
LSTM.MI	•	•	•	•	•	•	•	–

Table 9
The evaluation time (in ms) of CS-NN, CS-SVM, CS-C4.5, WELM, HMM, C5.0, LSTM and LSTM.MI.

	CS-NN	CS-SVM	CS-C4.5	WELM	HMM	C5.0	LSTM	LSTM.MI
Evaluation time (ms)	102	100	98	N/A	48	95	9	18

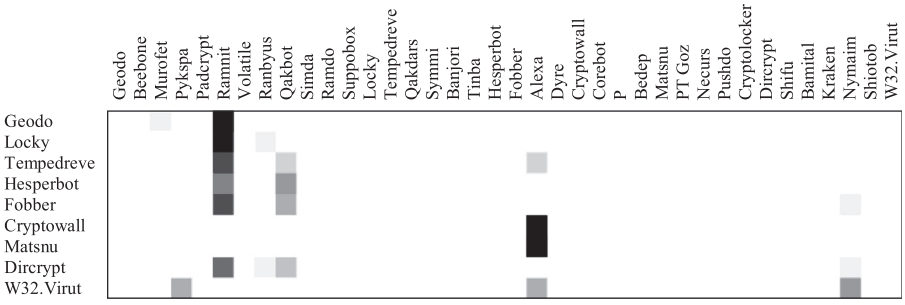

Fig. 6. Confusion matrix related to the malware families that cannot be detected by the LSTM.MI algorithm. The values are normalized to be in the range [0, 1]. 1 is illustrated as black, while 0 is illustrated as white.

Table 10
Summary of the real-world collected datasets.

	DGA type	# DNS Queries	# Malicious Domains
CTU-Malware-Capture-Botnet-142-1	Shifu	1085,080	1552
CTU-Malware-Capture-Botnet-150-1	Tinba	686,400	1002
CTU-Malware-Capture-Botnet-170-1	Necurs	5459	2709
CTU-Malware-Capture-Botnet-183-1	Locky	442	12

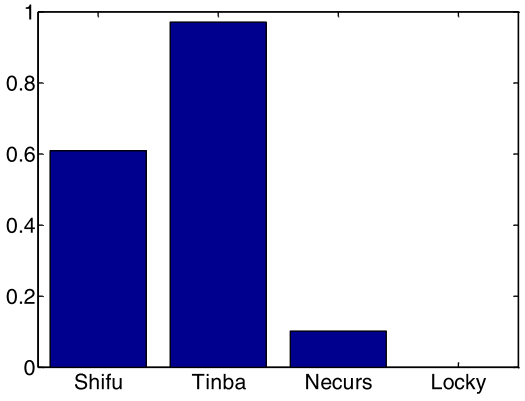

Fig. 7. The recall obtained on real-world collected datasets.

hand-crafted features to be extracted. Overall, CS-NN, CS-SVM, CS-C4.5, HMM and C5.0 are not well suited to deploy in real-time DGA detection applications.

There are still 9 DGA families that cannot be detected by LSTM.MI. As it can be seen in Fig. 6, *Geodo, Locky, Tempedreve, Hesperbot, Fobber* and *Dircrypt* are misclassified as *Ramnit* due to the fact that these malwares have the generator, which provides uniform distribution over the letters [3]. *Cryptowall* and *Matsnu* are based on pronounceable domains that cannot be easily isolated from the *Alexa* (non-DGA) class. For this reason, the macro-averaging F1-score in Table 6 is not as high as that in Table 4. C5.0 is able to accurately classify *Tempedreve* and *Locky*. It however cannot recognize *Ramnit*. Since *Ramnit* is a majority class, C5.0 is observed to be inferior to both the LSTM and LSTM.MI.

Finally, we evaluate the LSTM.MI on real-world collected datasets [55] that involve 4 malware families, namely *Shifu, Tinba, Necurs*, and *Locky*. Each dataset contains malware, normal and background traffic. The normal traffic is essential to compute the True Negative (TN) and False Positive (FP), while the background traffic is necessary to saturate the algorithm [55]. We run a script to execute DNS queries. A domain is labeled as being malicious if its query results in Non-Existent Domain (NXDOMAIN). Table 10 summarizes the properties of the datasets: the total number of DNS queries (# DNS Queries), number of malicious domains (# Malicious Domains). The precision is omitted due to the difference in the number of non-DGA domains. From Fig. 7, the obtained recall is similar to the recall in Table 6. This confirms the observations achieved in this paper and implies that the LSTM.MI is able to produce a high accuracy in practical applications.

5. Conclusions

This paper presents a novel LSTM.MI algorithm to handle the multiclass imbalance problem in DGA botnet detection. This method is based on both the binary and multiclass classification models, where the original LSTM is adapted to be cost-sensitive. The cost items are class-dependent, which take into consideration classification importance between classes. They are directly introduced into the backpropagation learning mechanism and the cost ratios are controlled by the γ parameter. We show that changing γ would result in a change in the decision boundary. The phenomena can be explained as resampling the training dataset.

The experiments demonstrate that the cost-sensitive LSTM is better alternative to the cost-insensitive one. It is also observed to be superior to RUSBoost, oversampling and Threshold-moving methods. These observations provide valuable insight to analyze the various characteristics of the LSTM.MI algorithm, which is able to achieve much higher macro-averaging F1-score with respect to HMM, C5.0, LSTM, the cost-sensitive SVM, cost-sensitive C4.5 and Weighted Extreme Learning Machine on the multiclass imbalanced dataset. This technique shares some important features with LSTM, making it amenable to immediate applications.

Like all other models, LSTM.MI cannot recognize *Cryptowall*, while the detection rates on *Matsnu* and *Suppobox* are relatively low. We note that *Cryptowall, Matsnu*, and *Suppobox* rely on pronounceable domains; identifying them in practice is extremely difficult without using contextual information [3]. Future research is therefore focused on investigating these malware families to further improve the overall system accuracy.

Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve this paper.

This work is supported by the Ministry of Science and Technology of Vietnam (MOST) under the grant no. KC.01.01/16-20.

References

[1] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon, From throw-away traffic to bots: detecting the rise of DGA-based Malware. In: Proceedings of the Twenty-first USENIX Security Symposium (USENIX Security 12) (2012).

[2] S. Schiavoni, F. Maggi, L. Cavallaro, S. Zanero, Phoenix, DGA-based botnet tracking and intelligence, in: Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA), in: Lecture Notes in Computer Science, 8550, 2014, pp. 192–211.

[3] J. Woodbridge, H.S. Anderson, A. Ahuja, and D. Grant, Predicting domain generation algorithms with long short-term memory networks. CoRR abs/1611.00791 (2016). arXiv:1611.00791.

[4] M. Kuhrer, C. Rossow, T. Holz, Paint it black: evaluating the effectiveness of malware blacklists,, in: Proceedings of the Research in Attacks, Intrusions and Defenses (RAID), Gothenburg, Sweden, 2014, pp. 1–21.

[5] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.

[6] F.A. Gers, J. Schmidhuber, F. Cummins, Learning to forget: continual prediction with LSTM, Neural Comput. 12 (10) (2000) 2451–2471.

[7] Y. Sun, M.S. Kamel, A.K. Wong, Y. Wang, Cost-sensitive boosting for classification of imbalanced data, Pattern Recogn. 40 (12) (2007) 3358–3378.

[8] S. Wang, X. Yao, Multiclass imbalance problems: Analysis and potential solutions, IEEE Trans. Syst. Man Cybern Part B Cybern. 42 (4) (2012) 1119–1130.

[9] Z.H. Zhou, X.Y. Liu, Training cost-sensitive neural networks with methods addressing the class imbalance problem, IEEE Trans. Knowl. Data Eng. 18 (1) (2006) 63–77.

[10] J. Hagen, and S. Luo, Why domain generation algorithms (DGA)? Trend Micro (Posted on: August 18, 2016).

[11] S. Yadav, A.K.K. Reddy, A.L. Reddy, S. Ranjan, Detecting algorithmically generated malicious domain names, in: Proceedings of the Tenth ACM SIGCOMM Conference on Internet measurement, 2010, pp. 48–61.

[12] R. Perdisci, I. Corona, G. Giacinto, Early detection of malicious flux networks via large-scale passive DNS analysis, IEEE Trans. Dependable Secure Comput. 9 (5) (2012) 714–726.

[13] L. Bilge, E. Kirda, C. Kruegel, M. Balduzzi, EXPOSURE: finding malicious domains using passive DNS analysis, in: Proceedings of the Network and Distributed System Security Symposium (NDSS), 2011.

[14] D.K. McGrath, M. Gupta, Behind Phishing, An examination of phisher Modi operandi, in: Proceedings of the First USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET), 2008.

[15] J. Ma, L.K. Saul, S. Savage, and G. Voelker, Beyond blacklists: Learning to detect malicious Web sites from suspicious URLs. In: Proceedings of the Knowledge discovery and data mining ACM KDD (2009).

[16] S. Yadav, A.K.K. Reddy, A.N. Reddy, S. Ranjan, Detecting algorithmically generated domain-flux attacks with DNS traffic analysis, IEEE/ACM Trans. Netw. 20 (5) (2012) 1663–1677.

[17] R.V. Salomon, J.C. Brustoloni, Identifying botnets using anomaly detection techniques applied to DNS traffic, in: Proceedings of the Fifth IEEE Consumer Communications and Networking Conference (CNCC), 2008, pp. 476–481.

[18] J. Kwon, J. Lee, H. Lee, A. Perrig, PsyBoG: a scalable botnet detection method for large-scale DNS traffic, Comput. Netw. 97 (2016) 48–73.

[19] G. Gu, R. Perdisci, J. Zhang, W. Lee, BotMiner, Clustering analysis of network traffic for protocol-and structure-independent botnet detection, USENIX Secur. Symp. 5 (2) (2008) 139–154.

[20] S. Krishnan, T. Taylor, F. Monrose, J. McHugh, Crossing the threshold: detecting network malfeasance via sequential hypothesis testing, in: Proceedings of the Fourty-third Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2013, pp. 1–12.

[21] F.A. Gers, N.N. Schraudolph, J. Schmidhuber, Learning precise timing with LSTM recurrent networks, J. Mach. Learn. Res. 3 (2002) 115–143.

[22] T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, S. Khudanpur, Recurrent neural network based language model, Interspeech 2 (3) (2010).

[23] A. Graves, A.R. Mohamed, G. Hinton, Speech recognition with deep recurrent neural networks, IEEE International Conference on Acoustics, Speech And Signal Processing (ICASSP), 2013.

[24] C.X. Ling, S.S. Victor, in: Cost-Sensitive Learning, Encyclopedia of Machine Learning, Springer, US, 2011, pp. 231–235.

[25] L. Jiang, C. Qiu, C. Li, A novel minority cloning technique for cost-sensitive learning, Int. J. Pattern Recogn. Artif. Intell. 29 (4) (2015) 1551004.

[26] C. Qiu, L. Jiang, G. Kong, A differential evolution-based method for class-imbalanced cost-sensitive learning, in: Proceedings of 2015 International Joint Conference on Neural Network, 2015, pp. 1–8.

[27] C. Elkan, The foundations of cost-senstive learning, in: Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, Seattle, Washington, 2011, pp. 973–978.

[28] Y. Liu, H. Lu, K. Yan, H. Xia, C. An, Applying cost-sensitive extreme learning machine and dissimilarity integration to gene expression data classification, Comput. Intell. Neurosci. (2016).

[29] K. Yan, Z. Ji, H. Lu, J. Huang, W. Shen, Y. Xue, Fast and accurate classification of time series data using extended ELM: application in fault diagnosis of air handling units, IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2017.

[30] K.M. Ting, An instance-weighting method to induce cost-sensitive trees, IEEE Trans. Knowl. Data Eng. 14 (3) (2002) 659–665.

[31] S. Zhang, X. Zhu, J. Zhang, C. Zhang, Cost-time sensitive decision tree with missing values, in: Proceedings of the Knowledge Science, Engineering and Management, 4798, 2007, pp. 447–459.

[32] H. Lu, L. Yang, K. Yan, Y. Xue, Z. Gao, A cost-sensitive rotation forest algorithm for gene expression data classification, Neurocomputing 228 (2017) 270–276.

[33] L. Jiang, C. Li, S. Wang, Cost-sensitive Bayesian network classifiers, Pattern Recogn. Lett. 45 (2014) 211–216.

[34] G. Wu, E.Y. Chang, Adaptive feature-space conformal transformation for imbalanced data learning, in: Proceedings of the Twentieth International Conference on Machine Learning, 2003, pp. 816–823.

[35] W. Zong, G.-B. Huang, Y. Chen, Weighted extreme learning machine for imbalance learning, Neurocomputing 101 (2013) 229–242.

[36] Z.H. Zhou, L. Xu-Ying, On multi-class cost-sensitive learning, Comput. Intell. 26 (3) (2010) 232–257.

[37] H. He, E.A. Garcia, Learning from imbalanced data, IEEE Trans. Knowl. Data Eng. 21 (9) (2009) 1263–1284.

[38] X.Y. Liu, J. Wu, Z.H. Zhou, Exploratory undersampling for class-imbalance learning, IEEE Trans. Syst. Man Cybern. Part B Cybern. 39 (2) (2009) 539–550.

[39] Y. Freund, R.E. Schapire, Experiments with a new boosting algorithm, in: Proceedings of the International Conference on Machine Learning (ICML), 96, 1996, pp. 148–156.

[40] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, F. Herrera, A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches, IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. 42 (4) (2012) 463–484.

[41] C. Seiffert, T.M. Khoshgoftaar, J. Van Hulse, A. Napolitano, RUSBoost, A hybrid approach to alleviating class imbalance, IEEE Trans. Syst. Man Cybern. Part A Syst. Human 40 (1) (2010) 185–197.

[42] N.V. Chawla, A. Lazarevic, L.O. Hall, K.W. Bowyer, SMOTEBoost, Improving prediction of the minority class in boosting, in: Knowledge Discovery in Databases: PKDD, Springer, Berlin Heidelberg, 2003, pp. 107–119. (2003).

[43] Q.D. Tran, P. Liatsis, RABOC, An approach to handle class imbalance in multimodal biometric authentication, Neurocomputing 188 (2016) 167–177.

[44] M. Kukar, I. Kononenko, Cost-sensitive learning with neural networks, in: Proceedings of the Thirteenth European Conference on Artificial Intelligence (ECAI), 1998, pp. 445–449.

[45] L.V.D. Maaten, G. Hinton, Visualizing data using t-SNE, J. Mach. Learn. Res. 9 (2008) 2579–2605.

[46] F. CholletKeras, 2016. https://github.com/fchollet/keras.

[47] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: machine learning in python, J. Mach. Learn. Res. 12 (2011) 2825–2830.

[48] Does Alexa have a list of its top-ranked websites? 2016. Available online at: https://support.alexa.com/hc/en-us/articles/200449834-Does-Alexa-have-a-list-of-its-top-ranked-websites-.

[49] Bambenek Consulting - Master feeds, 2016. Available online at: http://osint.bambenekconsulting.com/feeds/.

[50] M. Sokolova, G. Lapalme, A systematic analysis of performance measures for classification tasks, Inf. Process. Manag. 45 (4) (2009) 427–437.

[51] A. Graves, J. Schmidhuber, Framewise phoneme classification with bidirectional LSTM and other neural network architectures, Neural Netw. 18 (5) (2005) 602–610.

[52] R. Quinlan, Data mining tools see5 and c5, 2008. Available online at: https://www.rulequest.com/.

[53] J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera, KEEL data-mining software tool: data set repository, integration of algorithms and experimental analysis framework, J. Multiple Valued Logic Soft Comput. 17 (2–3) (2011) 255–287.

[54] L. Jiang, S. Wang, S.C. Li, L. Zhang, Structure extended multinomial naive Bayes, Inf. Sci. 329 (2016) 346–356.

[55] S. Garcia, The stratosphere IPS project, 2015. Available online at https://stratosphereips.org/category/dataset.html.

Duc Tran received the M.Sc. degree in electrical engineering from Budapest University of Technology and Economics, Hungary, in 2008, and the Ph.D. degree in information engineering from City University London, United Kingdom, in 2015. He is currently a research fellow with Bach Khoa Cybersecurity Centre, Hanoi University of Science and Technology, Vietnam. His research interests include machine learning, image processing, biometric authentication, and various pattern recognition-related algorithms.

Hieu Mac received the M.Sc. degree in data communication and computer networks from the Hanoi University of Science and Technology, Vietnam, in 2016, where he is currently pursuing the Ph.D. degree with Bach Khoa Cybersecurity Centre. His research interests include feature learning and extraction, machine learning, and applications in computer security.

Van Tong received the B.Eng. degree in computer science from Hanoi University of Science and Technology, Vietnam, in 2017. His current research interests include deep learning, network security and IoT.

Hai Anh Tran is a Lecturer in the School of Information and Communication Technology, Hanoi University of Science and Technology, Vietnam. He completed his Ph.D. at University of Paris-Est Creteil (France) (UPEC) in 2012. His research interests lie in the area of Distributed Systems, QoE, Data Mining, IoT, ranging from theory to design to implementation. He has actively collaborated with researchers in several other disciplines of computer science, particularly security and cryptography. Tran has served on conference and workshop program committees for SOICT conference since 2015.

Linh Giang Nguyen is an associate professor at the Department of Data Communication and Computer Networks, Hanoi University of Science and Technology, Vietnam. He received his M.Eng. degree in 1991 and Ph.D. degree in 1995 in conputer engineering from the Georgian University of Technology, Georgia, former USSR. His research interests include machine learning, data analysis, optimization, wireless sensor networks, IoT, network security, signal processing.