World Wide Web (2022) 25:401–423
https://doi.org/10.1007/s11280-021-00909-z

Vulnerability exploitation time prediction: an integrated framework for dynamic imbalanced learning

Jiao Yin[1,2] · MingJian Tang[3] · Jinli Cao[1] · Hua Wang[4] · Mingshan You[4] · Yongzheng Lin[5]

Received: 16 December 2020 / Revised: 27 April 2021 / Accepted: 4 June 2021 /
Published online: 1 July 2021

Abstract
Exploitation time is an essential factor for vulnerability assessment in cybersecurity management. In this work, we propose an integrated consecutive batch learning framework to predict the probable exploitation time of vulnerabilities. To achieve a better performance, we combine features extracted from both vulnerability descriptions and the Common Vulnerability Scoring System in the proposed framework. In particular, we design an Adaptive Sliding Window Weighted Learning (ASWWL) algorithm to tackle the dynamic multiclass imbalance problem existing in many industrial applications including exploitation time prediction. A series of experiments are carried out on a real-world dataset, containing 24,413 exploited vulnerabilities disclosed between 1990 and 2020. Experimental results demonstrate the proposed ASWWL algorithm can significantly enhance the performance of the minority classes without compromising the performance of the majority class. Besides, the proposed framework achieves the most robust and state-of-the-art performance compared with the other five consecutive batch learning algorithms.

Keywords Exploitation time prediction · Consecutive batch learning · Imbalanced learning · Multiclass learning

This article belongs to the Topical Collection: *Special Issue on Web Information Systems Engineering 2020*
Guest Editors: Hua Wang, Zhisheng Huang, and Wouter Beek

✉ Jiao Yin
j.yin@latrobe.edu.au

[1] Department of Computer Science and Information Technology, La Trobe University, Melbourne, VIC 3083, Australia

[2] School of Artificial Intelligence, Chongqing University of Arts and Sciences, Chongqing 402160, China

[3] Huawei Technologies Co. Ltd, Shenzhen 518129, China

[4] Institute for Sustainable Industries & Liveable Cities, Victoria University, Melbourne, VIC 3083, Australia

[5] Provincial Key Laboratory of Network Based Intelligent Computing, University of Jinan, Jinan 250022, China

1 Introduction

Background An increasing number of cybersecurity vulnerabilities are discovered and published each year. These existing weak spots pose severe threats to the security of millions of information systems [27, 33]. Practitioners in cybersecurity management are struggling in prioritizing and patching the countless disclosed vulnerabilities, due to relatively limited investment in time, workforce and resources [1, 17, 29]. Therefore, they have to make a trade-off between achieving the greatest coverage (patching as many vulnerabilities as they can to ensure security) and the highest efficiency (patching as fewer vulnerabilities as they can to save resources) [12]. For example, it should be a priority to patch a vulnerability that could be fixed within 15 minutes but may cost $1 million if leveraged, rather than to solve the one that would take hours to remedy but would not affect any critical systems if attacked. Obviously, reliable vulnerability assessment can provide valuable information for decision-making and therefore is of importance for cybersecurity management [23, 28]. However, this field still remains immature despite significant innovations and progresses over the past 20 years in both industry and academia [2, 24, 25].

Literature review The practice and research on vulnerability assessment have started for a long time [21, 30]. In industry, the most comprehensive and widely-used method is the Common Vulnerability Scoring System[1] (CVSS), which provides a score between 0-10 to indicate the risk of a vulnerability, according to dozens of attributes, including required privileges, exploitability score, attack complexity, integrity impact, et al. However, CVSS has long been criticized for being inaccurate to indicate the exploitability of vulnerabilities [5, 35, 36], which weakens its reliability and credibility.

To overcome this deficiency, a substantial number of previous works have been trying to develop more accurate and reliable exploitability prediction models, leveraging the latest machine learning and deep learning models as well as the rich historical vulnerability and exploit information [38, 39]. For example, Paper [26] proposed a neural language model-based approach, named DarkEmbed, to predict whether vulnerabilities will be exploited or not. Paper [8] investigated the effectiveness of different features, including common words from vulnerability descriptions, external references and vendor products, CVSS scores and categorical attributes, Common Weakness Enumeration (CWE) numbers, in predicting the exploitability of vulnerabilities. Jay Jacobs, Sasha Romanosky et al. [13] proposed an open, data-driven framework, called Exploit Prediction Scoring System (EPSS), to estimate the probability of a vulnerability being exploited within the first twelve months after disclosure. Jiao Yin, Mingjian Tang, et al. [35] employed transfer learning to extract paragraph-level embeddings from vulnerabilities and built a high-performance exploitation predictive model.

Although previous efforts have achieved significant improvements in exploitability assessment compared to CVSS, they all belong to binary classification algorithms and thus are incapable of predicting exploitation time in finer granularity. Besides, previous studies handled historical data in a batch learning manner, ignoring the temporal characteristics and possible implicit concept drifts in the data. Previous studies also overlooked another challenging problem, dynamic imbalanced data, by adopting batch learning.

[1] https://www.first.org/cvss/

Contributions In response to the existing problems, we propose an integrated consecutive batch learning framework to predict the exploitation time in finer granularity. Instead of dealing data with batch learning, we adopt consecutive batch learning as the online learning strategy to enable the classifier to capture the real-time temporal concepts. Furthermore, we design an Adaptive Sliding Window Weighted Learning (ASWWL) algorithm to tackle the dynamic class imbalance problem caused by online learning to achieve better performance.

To summarize, our main contributions are as follows:

(1) We propose an integrated consecutive batch learning framework to predict the exploitation time of vulnerabilities. The proposed framework handles historical data in a consecutive batch learning manner, enabling the classifier to capture the temporal features and possible concept drifts caused by the evolving system and user behaviours. Compared with previous studies, we also predict the exploitation time as a multiclass classification problem instead of a binary classification problem to provide more fine-grained results for decision-makers.

(2) We design a general learning algorithm, ASWWL, to deal with the dynamic multi-class imbalance problem, which is pervasive in real-world applications. We demonstrate the effectiveness of ASWWL in improving the performance of the minority classes, by a series of experiments conducted on a real-world dataset.

(3) The proposed integrated consecutive batch learning framework with the ASWWL algorithm achieves the most robust and state-of-the-art performance on the multiclass exploitation time prediction task, compared with the other five consecutive batch learning algorithms on the same dataset.

This paper is an extension of our previous work [36]. We extend techniques and devise a complete architecture for multiclass exploitation time prediction task. The improved framework is presented, and comprehensive experimental results are evaluated. The rest of this paper is organized as follows. Section 2 reviews some closely related works, followed by a detailed description of the proposed integrated framework and the ASWWL algorithm in Section 3. The datasets, evaluation metrics and experimental results are presented in Section 4. Section 5 concludes this paper with a discussion on limitations and future work.

2 Related work

2.1 Learning strategy

According to the availability of data and the way of feeding data to train and test classifiers, there are two learning algorithms, i.e. batch learning and stream learning. Batch learning is used for applications in which all data is available [7, 11, 22]. In this case, classifiers are usually evaluated in a hold-out manner, which means that data is randomly divided into a training set and a test set [34, 40] and classifiers are evaluated in a separated test set [18, 37]. The data imbalanced status is determined and static in batch learning because all data is available during the training process.

By contrast, for stream learning, data appears over time and classifiers are incrementally trained when new data is available. In this case, classifiers are often evaluated on the newly arrived data before using it to train the classifiers [20]. Stream learning inherently faces more challenges than batch learning. For example, concept drifts and dynamic class imbalance are two major problems that need to be addressed.

Stream learning has been studied for a long time [16], among them ensemble learning is the most popular strategy to boost performance. Back in 2001, a Concept-adapting Very

Fast Decision Tree (CVFDT) was proposed to learn time-changing concepts by keeping a decision tree consistent with a window of examples, based on a batch learning version of Very Fast Decision Tree (VFDT) [10]. In 2007, Kolter, J. Z. and Maloof, A. M. proposed a Dynamic Weighted Majority (DWM) ensemble algorithm, which dynamically trained multiple learners and weighted them according to their performance to get a global ensemble result [14]. The authors concluded that DWM outperformed algorithms that employ incrementally learning with a single learner, train all previously observed examples, or employ an unweighted, fixed-size ensemble of learners [14]. Paper [4] proposed a Hoeffding Tree Adaptive (HTA) algorithm to adaptively learn concept drifts over time, by maintaining a Hoeffding Window Tree and a Hoeffding Adaptive Tree at the same time in 2009. To handle the concept drift in nonstationary environments (NSEs), paper [9] introduced an ensemble Learn^{++}.NSE (LPPNSE) algorithm in 2011, by employing a dynamically weighted majority voting mechanism. Kosina, P. and Gama, J developed a Very Fast Decision Rules Classifier (VFDRC)[15] and the adaptive extension (AVFDRC) to detect and adapt changes in data distributions in 2015. Paper [19] proposed a Self Adjusting Memory K Nearest Neighbor (SAMKNN) model for heterogeneous concept drift, inspired by biological memory models in 2016. An open-source framework, named Scikit-Multiflow[2] was released in 2018, providing the implementation of multiple steam data generators, stream learning algorithms, evaluators and transformers [20]. In 2019, Anwar, M. M. et al. proposed an index-based algorithm to discover and track the evolution of user groups drove by time-sensitive activities in social networks [3].

2.2 Multiclass imbalanced learning

Parts of data streams suffer from dynamic class imbalance, which severely damages the performance of classifiers. To solve this problem, paper [31] designed a formula (1) to calculate the Imbalance Factor (IF) of each single class $c_{[k]}$ at time step t.

$$w_{[k]}^{(t)} = \frac{(t-1) * w_{[k]}^{(t-1)} + [(x^{(t)}, c_{[k]})]}{t}, (k = 1, 2, \cdots, n_c \text{ and } t \geq 1) \tag{1}$$

where $w_{[k]}^{(t)}$ is the Imbalance Factor of class $c_{[k]}$ at time step t and $w_{[k]}^{(0)}$ is initialized as 0; n_c is the total number of class labels; $x^{(t)} \in \mathbb{R}^n$ is a feature vector in a n dimensional space $\mathcal{X}$ observed at time step t; $[(x^{(t)}, c_{[k]})]$=1 if the true label of $x^{(t)}$ equals to $c_{[k]}$, otherwise 0.

Equation (1) actually calculates the global Imbalance Factor over all previously encountered examples until time step t. As t increases, $w_{[k]}^{(t)}$ would be less sensitive to the latest imbalanced status.

An improvement of (1) in weakening the influence of older data is introduced in paper [32]. As shown in formula (2), $w_{[k]}^{(t)}$ is updated along with a time decay factor θ.

$$w_{[k]}^{(t)} = \theta w_{[k]}^{(t-1)} + (1-\theta)[(x^{(t)}, c_{[k]})], (k = 1, 2, \cdots, n_c \text{ and } t \geq 1) \tag{2}$$

Equation (2) is criticized as making a too strong assumption that all classes decay at the same rate θ [36]. Therefore, the authors in paper [36] designed a Sliding Window Imbalance

[2] https://github.com/scikit-multiflow/scikit-multiflow

Factor (SWIF) to present the latest class proportions in a data stream, as shown in (3).

$$w_{[k]}^{(t)} = \begin{cases} \frac{(t-1)*w_{[k]}^{(t-1)}+[(x^{(t)},c_{[k]})]}{t}, & (k = 1, 2, \cdots, n_c \text{ and } 1 \leq t < z) \\ \frac{z*w_{[k]}^{(t-1)}-[(x^{(t-z)},\ c_{[k]})]+[(x^{(t)},\ c_{[k]})]}{z}, & (k = 1, 2, \cdots, n_c \text{ and } t \geqslant z) \end{cases} \tag{3}$$

where z ($z \geq n_c$) is the number of samples within a sliding window, which could be fixed during the whole online learning process or it could also be adaptively adjusted according to some strategies.

3 Methodology

3.1 Consecutive batch leaning

Consecutive batch learning is a special case of data stream learning, where stream data is handled in consecutive batches. Let $S = \{D^{(0)}, D^{(1)}, \cdots, D^{(k)}, \cdots\}$ be a chronological data stream, where $D^{(k)}=\{X^{(k)}, Y^{(k)}\}$ ($k \geq 1$) is the k-th batch of observed samples or instances. $X^{(k)}=\{x_1^{(k)}, x_2^{(k)}, \cdots, x_{n_b}^{(k)}\}$, where n_b is the total number of samples in the k-th data batch and $x_i^{(k)} \in \mathbb{R}^n$ (i= 1, 2, $\cdots$, n_b) is the feature vector of the i-th sample in the k-th batch. $Y^{(k)}=\{y_1^{(k)}, y_2^{(k)}, \cdots, y_{n_b}^{(k)}\}$ is the corresponding labels for batch k.

Let $f^{(1)}(\cdot), \cdots, f^{(k)}(\cdot), \cdots$ denote the consecutive classifier statuses which are incrementally trained from the consecutive data batches. For consecutive batch learning, at each time step k ($k \geq 1$), $f^{(k)}(\cdot)$ will be used to predict the labels of arrived data $X^{(k)}$ for the usage of downstream applications, when the corresponding true labels $Y^{(k)}$ are unavailable. The predicted labels $\hat{Y}^{(k)}$ is calculated as (4).

$$\hat{Y}^{(k)} = f^{(k)}(X^{(k)}), (k \geq 1). \tag{4}$$

After the ground truth $Y^{(k)}$ for the k-th data batch is available, the labelled data $D^{(k)}=\{X^{(k)}, Y^{(k)}\}$ will be used to fit $f^{(k)}(\cdot)$ to get the next classifier status $f^{(k+1)}(\cdot)$.

3.2 An integrated framework for exploitation time prediction

The work flow of the proposed integrated consecutive batch leaning framework for exploitation time prediction problem is illustrated in Figure 1. The notations in Figure 1 are consistent with the definitions in Section 3.1. Like other consecutive batch learning frameworks, there are two major stages for each data batch, namely, prediction and classifier update. Particularly, for vulnerability exploitation time prediction problem, the prediction stage includes the process of feature extraction from descriptions and CVSS attributes, and feature reduction and fusion. To facilitate discussion, we first explain all of the framework components displayed in Figure 1.

Descriptions are a collection of vulnerability descriptions in the k-th batch. Specifically, a description is a brief paragraph containing information like the affected products, the

required privilege, the possible attack paths of a vulnerability. Descriptions are usually disclosed with vulnerabilities by some publicly available database, such as the National Vulnerability Database (NVD)[3] and the Common Vulnerabilities and Exposures database(CVE).[4] Descriptions should be further treated before being fed to a classifier.

Pre-trained BERT is a released Bidirectional Encoder Representations from Transformers (BERT) model [6]. BERT has proven to be an excellent Natural Language Processing (NLP) model which can extract semantic features from both words and sentences. In the proposed framework, we adopt a pre-trained BERT model to extract sentence-level semantic feature $X_{des}^{(k)}$ from Descriptions instead of training a BERT model from scratch, considering the limited scale of vulnerability description corpus.

CVSS V2.0 represents a set of attributes selected from the 2.0 version of CVSS metrics. Some of the attributes are not numerical. For example, the value range of the attribute 'Access Complexity' is {High, Medium, Low}.

Encoding is the method to encode all CVSS V2.0 attributes to numerical features $X_{cvss}^{(k)}$.

Feature reduction and fusion involves a feature reduction process and a feature fusion process. In the proposed framework, we first separately reduce the dimension of $X_{des}^{(k)}$ and $X_{cvss}^{(k)}$ to an acceptable value by applying feature reduction algorithms. Then, fuse the results to form the feature $X^{(k)}$ of the k-th batch via concatenating, averaging, ranking or other fusion strategies.

Downstream applications are the possible industry applications relying on the predicted labels $\hat{Y}^{(k)}$, such as the cybersecurity management system, which will use the predicted labels to make decisions on patching the corresponding vulnerability or not.

Exploit-DB [5] is an open-source exploit database, working as the data source of ground truth in our proposed framework.

Ground truth is the class labels corresponding to $X^{(k)}$, which could be encoded into $Y^{(k)}$, by one hot encoding, label encoding, custom binary encoding or other methods.

ASWWL is the proposed Adaptive Sliding Window Weighted Learning algorithm. Instead of weighting different learners, we weigh each sample based on real-time imbalanced status. The detailed description of ASWWL will be presented in Section 3.3.

Classifier fitting is the process of fitting the classifier $f^{(k)}(\cdot)$ to the k-th data batch $\{X^{(k)}, Y^{(k)}\}$ with weight $W^{(k)}$ based on a predefined cost function.

As shown in Algorithm 1, the inputs of the proposed framework include data sources like descriptions, CVSS V2.0 and Exploit-DB, a pre-trained BERT model, an initialized classifier $f^{(1)}(\cdot)$ and the size of data batch n_b. As time passes, the framework will generate a series of intermediate results, namely, the sequence of features extracted from descriptions

[3] https://nvd.nist.gov/vuln/data-feeds
[4] https://cve.mitre.org
[5] https://www.exploit-db.com/

$\mathcal{X}_{des}$= $\{X_{des}^{(1)}, \cdots, X_{des}^{(k)}, \cdots\}$, features extracted from descriptions CVSS 2.0 $\mathcal{X}_{cvss}$=$\{X_{cvss}^{(1)}, \cdots, X_{cvss}^{(k)}, \cdots\}$, fused features $\mathcal{X}$=$\{X^{(1)}, \cdots, X^{(k)}, \cdots\}$, true labels $\mathcal{Y}$=$\{Y^{(1)}, \cdots, Y^{(k)}, \cdots\}$ and classifier statuses $f^{(2)}(\cdot), \cdots, f^{(k+1)}(\cdot), \cdots$. The final valuable output for downstream applications is the sequence of predicted labels $\widehat{\mathcal{Y}}$=$\{\widehat{Y}^{(1)}, \cdots, \widehat{Y}^{(k)}, \cdots\}$.

Algorithm 1 An integrated framework for exploitation time prediction.

Input: Descriptions; CVSS V2.0; Exploit-DB; Pre-trained BERT; $f^{(1)}(\cdot)$, n_b.
Output: $\widehat{\mathcal{Y}}$=$\{\widehat{Y}^{(1)}, \cdots, \widehat{Y}^{(k)}, \cdots\}$.

1: **for** the k-th batch ($k \geq 1$) **do**
2: Feed the k-th batch of Descriptions to a pre-trained BERT model to extract features $X_{des}^{(k)}$.
3: Encode the k-th batch of CVSS V2.0 attributes to numerous features $X_{cvss}^{(k)}$.
4: Apply feature reduction algorithm to $X_{des}^{(k)}$ and $X_{cvss}^{(k)}$ separately and fuse the results to get the fused feature $X^{(k)}$.
5: Predict the labels $\hat{Y}^{(k)}$ of $X^{(k)}$ for downstream applications by equation (4).
6: From Exploit-DB data find out the ground truth and encode it to $Y^{(k)}$.
7: $D^{(k)}$=$\{X^{(k)}, Y^{(k)}\}$.
8: **if** ASWWL == True **then**
9: Run Algorithm 2 and get the returned $W^{(k)}$
10: **else**
11: $W^{(k)}$=None
12: **end if**
13: Update $f^{(k)}$ to $f^{(k+1)}$ based on $D^{(k)}$ and $W^{(k)}$
14: **end for**

For the k-th batch ($k \geq 1$), steps 2-5 specify the implementation of the prediction stage shown in Figure 1, while steps 6-13 are the implementation of the classifier update stage. ASWWL is an optional strategy for improving performance, as shown in steps 8-12. The framework without ASWWL is a baseline version and the framework with ASWWL is an improved version, providing adaptive weights for each single sample in $D^{(k)}$.

3.3 Adaptive sliding window weighted learning

Adaptive Sliding Window Weighted Learning is designed for handling dynamic multiclass imbalance problem existing in data stream. Sliding window is a widely-used strategy in multiple areas [4, 10], such as signal processing and ensemble learning. We employ this idea to adaptively measure the current class imbalanced status and then calculate weights $W^{(k)}$=$\{w_1^{(k)}, w_2^{(k)}, \cdots, w_{n_b}^{(k)}\}$ for the k-th data batch.

Specifically, let $\mathbf{Y}_e^{(k)}$ denote the list of all existing sample labels in chronological order, and z ($z \geq n_c$) is the sliding window size, where n_c is the number of unique labels in the data stream. The first step of ASWWL is to find out the labels of the latest z samples, denoted as Y_{sw}. The second step is counting the number of samples N_c (c=1, 2, $\cdots$, n_c) belonging to each class within Y_{sw}. Finally, calculate the weight $w_i^{(k)}$ of the i-th sample in the k-th batch by formula (5).

$$w_i^{(k)} = \frac{N_{max}}{N}, (i = 1, 2, \cdots, n_b). \tag{5}$$

where $N_{max} = \max_{c=1}^{n_c} N_c$ and N is the number of samples belonging to the same class of sample $\{x_i^{(k)}, y_i^{(k)}\}$. Obviously, $w_i^{(k)} \geq 1$.

The implementation of ASWWL is illustrated in Algorithm 2. The input includes all existing true labels $\mathcal{Y}_e^{(k)}=\{Y^{(1)}, \cdots, Y^{(k)}\}$, the current batch number k, batch size n_b and a pre-set sliding window size z. The output is the weight for batch k. Step 1 initializes the weight for each sample in batch k to 1. Step 2 expands all existing label collections in $\mathcal{Y}_e^{(k)}$ into a list $\mathbf{Y}_e^{(k)}$. Steps 3-19 calculate the weight of each sample using a loop. Among them, steps 4-9 is to find out the labels of the latest samples within the sliding window Y_{sw}; steps 10-16 count the number of samples of each class in Y_{sw}; and steps 17-18 calculates the weight for the i-th sample $w_i^{(k)}$.

Algorithm 2 Adaptive sliding window weighted learning.

Input: $\mathcal{Y}_e^{(k)}=\{Y^{(1)}, \cdots, Y^{(k)}\}$, k, n_b, z.
Output: $W^{(k)}$.
1: $W^{(k)}$=ones(n_b,1)
2: $\mathbf{Y}_e^{(k)}$= [y for y in $\mathcal{Y}_e^{(k)}$]
3: **for** i in range(1,n_b) **do**
4: idx=$(k-1) * n_b + i$
5: **if** idx-$z \geq$0 **then**
6: Y_{sw}=$\mathbf{Y}_e^{(k)}$[inx-z:idx]
7: **else**
8: Y_{sw}=$\mathbf{Y}_e^{(k)}$[0:idx]
9: **end if**
10: C = unique(Y_{sw}), n_c=len(C).
11: **for** c in range(n_c) **do**
12: Count N_c (the number of samples belonging to $C[c]$ in Y_{sw}).
13: **if** $Y_i^{(k)}$==$C[c]$ **then**
14: $N = N_c$
15: **end if**
16: **end for**
17: $N_{max} = \max_{c=1}^{n_c} N_c$
18: $w_i^{(k)}$=N_{max}/N
19: **end for**
20: **return** $W^{(k)} = \{w_1^{(k)}, w_2^{(k)}, \cdots, w_{n_b}^{(k)}\}$

4 Experiments

In this section, we report the experimental results of exploitation time prediction. Section 4.1 presents the dataset and experimental setting of this work, followed by the introduction of evaluation metrics in Section 4.2. Section 4.3 gives the performance comparison of the baseline version and the ASWWL version of the proposed framework, adopting three

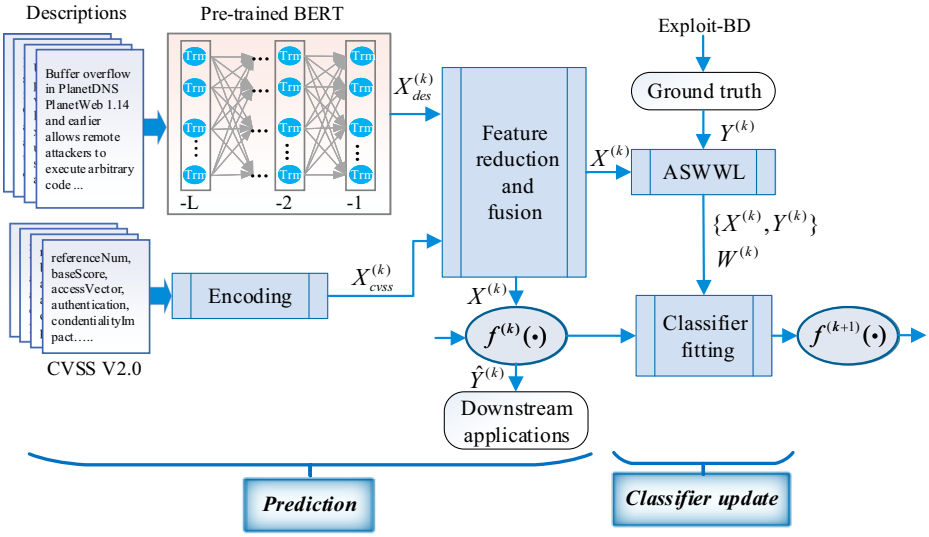

Figure 1 Workflow of the proposed integrated framework ($k \geq 1$)

classifiers, namely, Neural Networks (NN), Hoeffding Tree (HT) and Naive Bayes (NB). Section 4.4 provides the performance comparison of our method with other five data stream algorithms.

4.1 Dataset and experimental setting

The vulnerability data including descriptions and CVSS 2.0 attributes is collected from NVD and the exploit data comes from Exploit-DB. In this paper, when a 'proof-of-concept' exploit exists in Exploit-DB, the corresponding vulnerability is identified as 'exploited'. Vulnerabilities and exploits are integrated by CVE-IDs (a CVE-ID is a global identifica-

Table 1 Selected CVSS V2.0 attributes and their value types

Value types	CVSS 2.0 attributes
Numerical	Number of References*, Base Score, Impact Score, Exploitability Score
Categorical	Access Vector, Access Complexity, Authentication, Confidentiality Impact, Integrity Impact, Availability Impact, Severity
Boolean	User Interaction Required, Obtain User Privilege, acInsufInfo, Obtain Other Privilege

*This attribute does not belong to CVSS V2.0 metrics. It is the total number of reference information of a vulnerability listed in NVD dataset

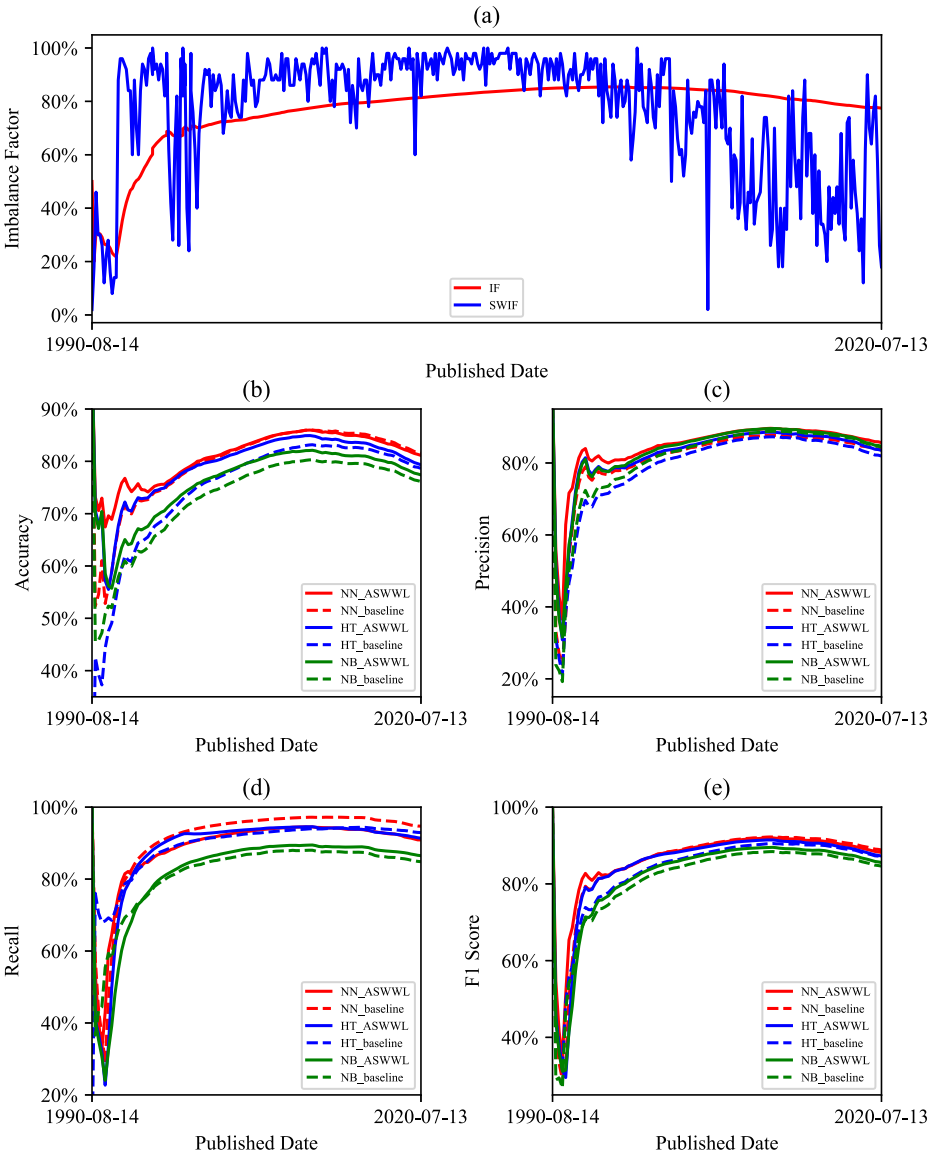

Figure 2 Realtime classification results on class 'Neg'

tion number for a vulnerability). The dataset used in this work contains 23,413 exploited vulnerabilities collected between 1990 to 2020. Among them, 18,127 (77.42%) vulnerabilities are labelled as 'Neg', which means that they are exploited before being disclosed; 3,971 (16.96%) vulnerabilities are exploited after being disclosed, labelled as 'Pos'; and the rest 1,315 (5.62%) vulnerabilities are exploited and disclosed at the same day, labelled as 'ZeroDay'.

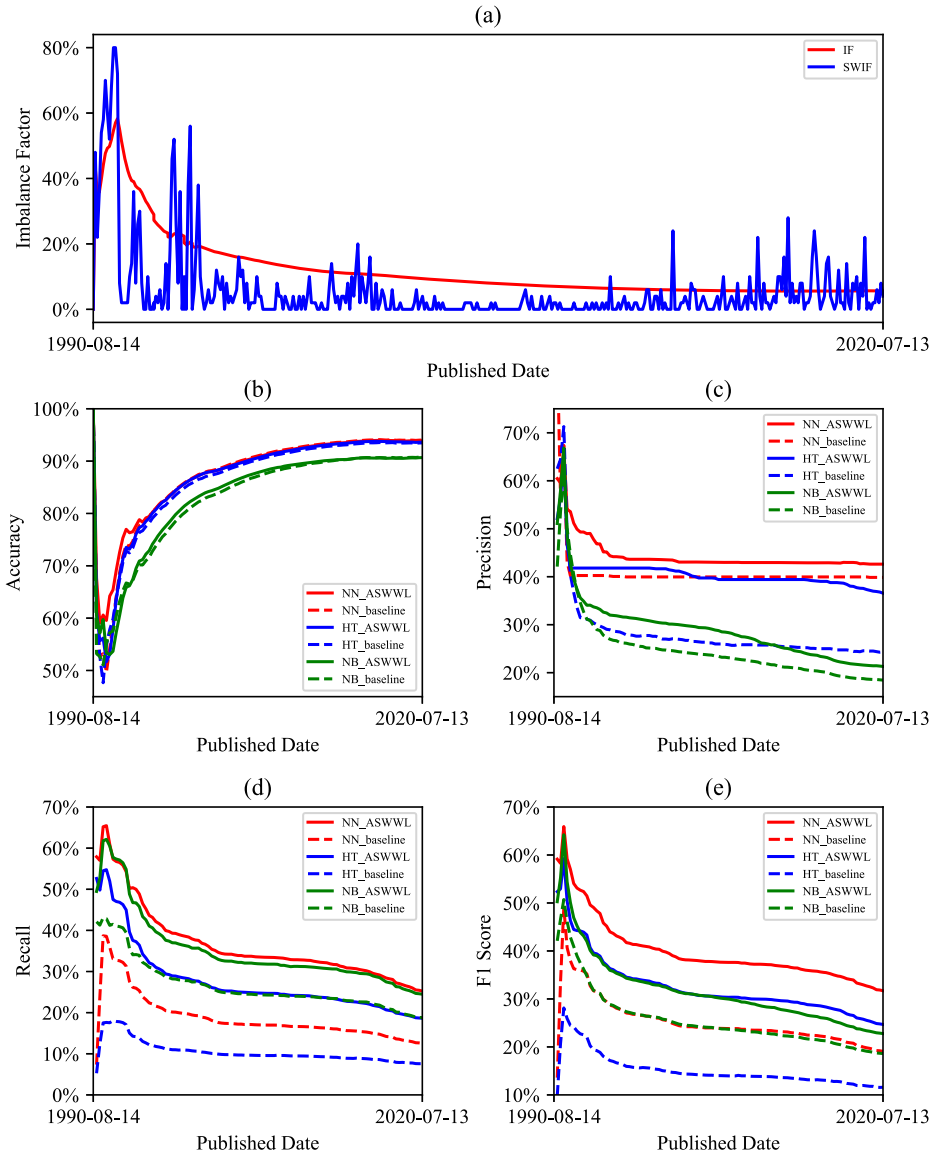

Figure 3 Realtime classification results on class 'ZeroDay'

The data batch size n_b is set to 100 for all experiments in this work. The uncased BERT-base version[6] of pre-trained BERT model is chosen to extract 768-dimensional features from descriptions, therefore $X_{des}^{(k)} \in \mathbb{R}^{768*100}$.

[6]https://storage.googleapis.com/bertmodels/2018_10_18/uncased_L-12_H-768_A-12.zip

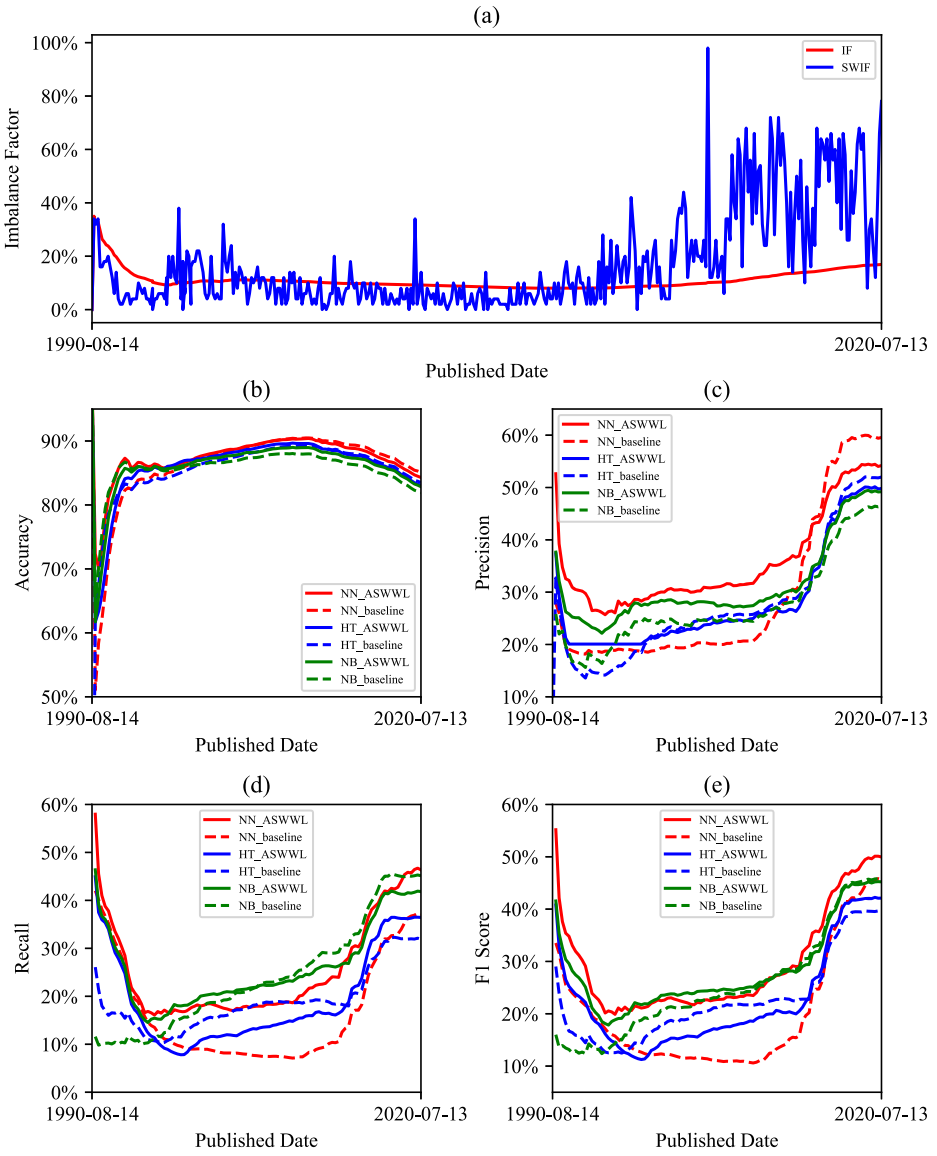

Figure 4 Realtime classification results on class 'pos'

Table 1 shows the selected CVSS V2.0 attributes and their value types in this work. All categorical attributes listed in Table 1 are encoded by one hot encoding method. After encoding, the features extracted from CVSS V2.0 $X_{cvss}^{(k)} \in \mathbb{R}^{30*100}$.

The feature reduction method adopted in this work is Principal Component Analysis (PCA) and the feature fusion method is concatenation. We extract 10 features from $X_{des}^{(k)}$ and $X_{cvss}^{(k)}$ respectively and then concatenate them to get a $X^{(k)} \in \mathbb{R}^{20*100}$.

Table 2 Overall classification results on all existing samples of class 'Neg'

Algorithms	Accuracy	Precision	Recall	F1 Score
NN_ASWWL	81.13%	85.64%	90.88%	88.18%
NN_baseline	81.57%	83.65%	94.71%	88.84%
HT_ASWWL	79.33%	83.50%	91.37%	87.26%
HT_baseline	78.65%	81.93%	92.93%	87.09%
NB_ASWWL	77.40%	84.66%	86.51%	85.57%
NB_baseline	76.14%	84.43%	84.83%	84.63%

4.2 Evaluation metrics

For classification problems, Accuracy, the number of correct predictions divided by all predictions made, is the most important evaluation metrics. However, for tasks with class imbalance, Accuracy alone can be misleading. Therefore, Precision and Recall are used as a supplement to show the exactness and completeness of a classifier. Furthermore, F1 Score is usually considered as the final measure to decide which classifier is better, because it conveys balances between Precision and Recall.

As exploitation time prediction is an imbalanced multiclass classification problem, we adopt Accuracy, Precision, Recall and F1 score to measure the performance of a classifier on each single class. Besides, we employ two average strategies to show the overall performance on all classes, namely, Macro and Micro.

Macro is an average strategy that calculates all metrics for each class separately, and then find their unweighted mean.

Micro is an average strategy that calculates all metrics globally on all classes by counting the total true positives, false positives, true negatives and false negatives.

4.3 Comparison between ASWWL and the baseline

Asxs described in Section 3.2, the proposed framework has a baseline version and an ASWWL version. To verify the effectiveness of the proposed ASWWL algorithm, we perform comparison studies on three different classifiers, namely, Neural Networks (NN), Hoeffding Tree (HT) and Naive Bayes (NB). The hyperparameters are set as follows, batch step n_b=100, sliding window size z=3 for classifiers NN and NB and z=5 for classifier HT.

In this subsection, we present the performance comparison on each single class separately in Figs. 2, 3, 4, and Tables 2, 3, 4, followed by the Macro and Micro average performance on all classes shown in Table 5.

4.3.1 Classification result on the majority class 'Neg'

Figure 2 shows the realtime classification result on class 'Neg. The red line, IF, in subplot (a) is the overall Imbalance Factor of class 'Neg', calculated by formula (2), while the blue line, SWIF, in subplot (a) represents the Sliding Window Imbalance Factor calculated by formula (3), when z=50. Obviously, SWIF can reflect the latest imbalanced status, while IF is much less sensitive than SWIF due to the influence of history samples.

Table 3 Overall classification results on all existing samples of class 'ZeroDay'

Algorithms	Accuracy	Precision	Recall	F1 Score
NN_ASWWL	93.89%	42.62%	25.25%	31.71%
NN_baseline	94.02%	39.86%	12.55%	19.09%
HT_ASWWL	93.61%	36.57%	18.63%	24.69%
HT_baseline	93.48%	24.33%	7.60%	11.59%
NB_ASWWL	90.67%	21.30%	24.49%	22.78%
NB_baseline	90.77%	18.45%	18.78%	18.61%

Subplots (b), (c), (d), (e) present four metrics respectively. Generally speaking, all these metrics fluctuate in the same trend as the Imbalance Factor of class 'Neg'. This is because, for all data-driven machine learning classifiers, a higher class proportion means better performance in the corresponding class. Even though ASWWL is proposed to reduce the influence of data imbalance, this trend still exists. Classifier NN, in red color, performs best, followed by HF in blue and NB in green. Besides, the ASWWL version performs better than the baseline version for all three classifiers most times.

Table 2 shows the quantitative results evaluated on all existing samples in a prequential-evaluation manner. Overall, NN_baseline achieves the best Accuracy, Recall and F1 Score, while NN_ASWWL performs the best in Precision. NN_ASWWL gets a slightly worse F1 Score than NN_baseline. However, the ASWWL version works better than the baseline version when adopting HT or NB as the classifier of the proposed framework.

Generally speaking, the difference between the ASWWL version and the baseline version on majority class 'Neg' is non-significant. This is because 'Neg' is the majority class, and ASWWL is proposed to boost the minority classes and further increase the overall performance. The experimental results in the following sections will show a significant increase in performance of the minority classes 'ZeroDay' and 'Pos', as well as the average performance over all classes, when applying ASWWL.

4.3.2 Classification result on the minority classes—'ZeroDay'

Similarly, Figure 3 shows the realtime classification results on one of the minority classes—'ZeroDay'. The subplot (a) shows the IF in red and SWIF in blue. Subplots (c), (d), (e) display the same fluctuation trend with (a).

Table 4 Overall classification results on all existing samples of class 'Pos'

Algorithms	Accuracy	Precision	Recall	F1 Score
NN_ASWWL	84.29%	54.19%	46.35%	49.97%
NN_baseline	85.09%	59.56%	37.05%	45.69%
HT_ASWWL	83.04%	49.83%	36.45%	42.10%
HT_baseline	83.50%	52.00%	31.90%	39.54%
NB_ASWWL	82.85%	49.20%	41.83%	45.22%
NB_baseline	81.91%	46.43%	45.14%	45.77%

Table 5 Average classification results on multiple classes

Average	Algorithms	Accuracy	Precision	Recall	F1 Score
Macro	NN_ASWWL	79.66%	60.82%	54.16%	56.62%
	NN_baseline	80.34%	61.02%	48.11%	51.20%
	HT_ASWWL	77.99%	56.63%	48.82%	51.35%
	HT_baseline	77.81%	52.75%	44.15%	46.07%
	NB_ASWWL	75.46%	51.72%	50.94%	51.19%
	NB_baseline	74.41%	49.77%	49.58%	49.67%
Micro	NN_ASWWL	79.66%	79.66%	79.66%	79.66%
	NN_baseline	80.34%	80.34%	80.34%	80.34%
	HT_ASWWL	77.99%	77.99%	77.99%	77.99%
	HT_baseline	77.81%	77.81%	77.81%	77.81%
	NB_ASWWL	75.46%	75.46%	75.46%	75.46%
	NB_baseline	74.41%	74.41%	74.41%	74.41%

For Accuracy in subplot (b), classifier NN and HT obtain an equivalent performance, better than classifier NB. There is little difference between the ASWWL version and the baseline version on Accuracy. However, the ASWWL version of all three classifiers perform significantly better than the baseline version on Precision, Recall and F1 Score, which verifies the effectiveness of ASWWL in boosting the performance of the minority classes.

Table 3 is the corresponding quantitative classification results evaluated on all existing samples. Obviously, the ASWWL version has an overwhelming advantage than the baseline version on Precision, Recall and F1 Score while keeping an equivalent performance on Accuracy. Classifier NN is also the best classifier among these three classifiers.

4.3.3 Classification result on the minority classes—'Pos'

Class 'Pos' is another minority class in exploitation time prediction problem. The realtime classification results on class 'Pos' are shown in Figure 4. Subplot (a) shows that in the beginning years, class 'Pos' accounted for about 35%, and then dropped to less than 20% very quickly. In recent years, its SWIF increased to a level of more than 60%, leading to a remarkable increase in the performance of Precision, Recall and F1 Score, as shown in subplot (c), (d), (e). Classifier NN also performs best in most times. Classifier NB performs much better than HT on Recall and F1 Score for class 'Pos'.

Table 4 presents the overall classification result on all existing samples of class 'Pos'. The results in Table 4 show that both baseline and ASWWL may achieve the best on some metrics. However, considering F1 Score is a balanced metrics reflecting both precision and recall, we conclude that the ASWWL version performs much better than the baseline version, when adopting NN and HT as the classifier. The baseline version performs slightly better than the ASWWL version when choosing NB as the classifier.

4.3.4 Average classification results on multiple classes

According to the description above, generally speaking, the ASWWL version performs better than the baseline version on F1 Score, especially for these two minority classes, no

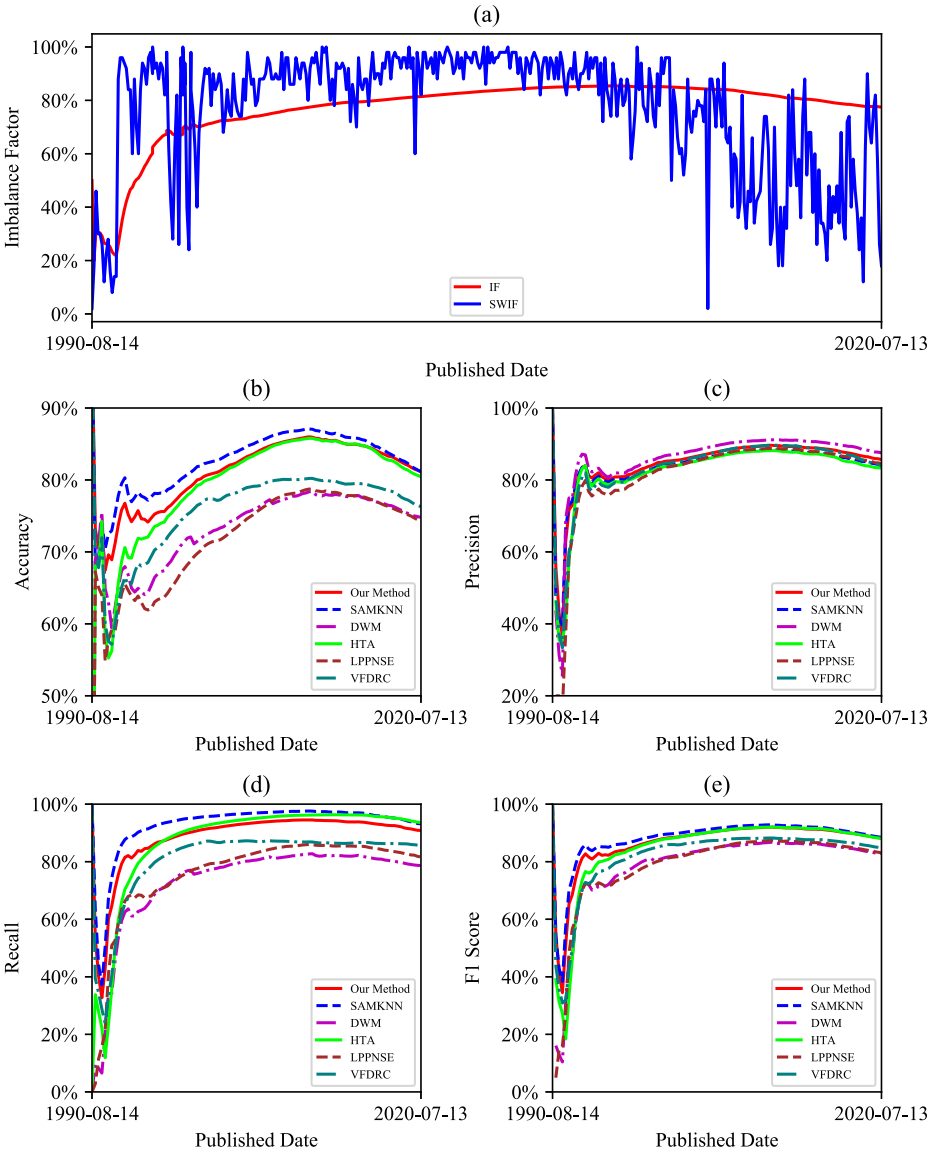

Figure 5 Realtime classification results on class 'Neg'

matter adopting which classifier. Two exceptions are NN_ASWWL on the class 'Neg' and NB_ASWWL on the class 'Pos'.

We list the Macro and Micro average performance over all three classes in Table 5. We can see that the ASWWL version achieves a much better Macro average F1 Score than the baseline version on all three classifiers. Among them, NN_ASWWL achieves the best Macro F1 Score on 56.62%, 5.42% higher than NN_baseline and 5.27% higher than the second place, HT_ASWWL. For Micro F1 Score, the ASWWL version with a HT or NB

Table 6 Overall classification result on all existing samples of class 'Neg'

Algorithms	Accuracy	Precision	Recall	F1 Score
Our Method	81.13%	85.64%	90.88%	88.18%
SAMKNN	81.15%	84.17%	93.19%	88.45%
DWM	74.80%	87.55%	78.65%	82.86%
HTA	80.44%	83.26%	93.57%	88.11%
LPPNSE	74.19%	84.50%	81.66%	83.06%
VFDRC	76.28%	83.98%	85.73%	84.85%

classifier also performs better than the baseline version over 1%. NN_baseline is slightly better than NN_ASWWL.

To conclude, ASWWL is effective to boost the classification performance of the minority classes, by giving larger weights to minority class samples adaptively.

4.4 Comparison with other data stream learning algorithms

The proposed framework is designed to predict the exploitation time in a data stream context. To verify its performance, we compared the ASWWL version of the proposed framework with a NN classifier (our method) with five data stream learning algorithms introduced in Section 2.1, namely, SAMKNN, DWN, HTA, LPPNSE and VFDRC on the same dataset. We first list the results on each class separately and then give the average performance on these classes.

4.4.1 Classification result on the majority class 'Neg'

The realtime classification performance comparison on class 'Neg' is shown in Figure 5. Subplot (a) is the realtime IF and SWIF of class 'Neg'. Generally speaking, SAMKNN, HTA and our method achieve better performance than the rest on all four metrics. The detailed results are summarized in Table 6.

As shown in Table 6, our method gets an 88.18% F1 Score, which is at the equivalent level of the best, 88.45%, achieved by SAMKNN. HTA also gets quite a high F1 Score at 88.11% compared with the rest. DWD performs the worst on class 'Neg', with an 82.86% F1 Score. our method also achieves almost the same Accuracy as SAMKNN. DWM performs best on Precision with 97.55% and HTA on Recall with 93.57%.

4.4.2 Classification result on the minority classes—'ZeroDay'

The realtime performance on one of the minority classes 'ZeroDay' is presented in Figure 6. Subplots (a) to (e) demonstrate the realtime imbalance status and four classification metrics on class 'ZeroDay'. SAMKNN presented by the blue dotted lines achieves the best Accuracy, Precision and F1 score, followed by our method in red solid lines. As for Recall, the HTA in green solid line achieves the best, followed by SAMKNN, our method and VFDRC. DWM performs the worst on class 'ZeroDay'.

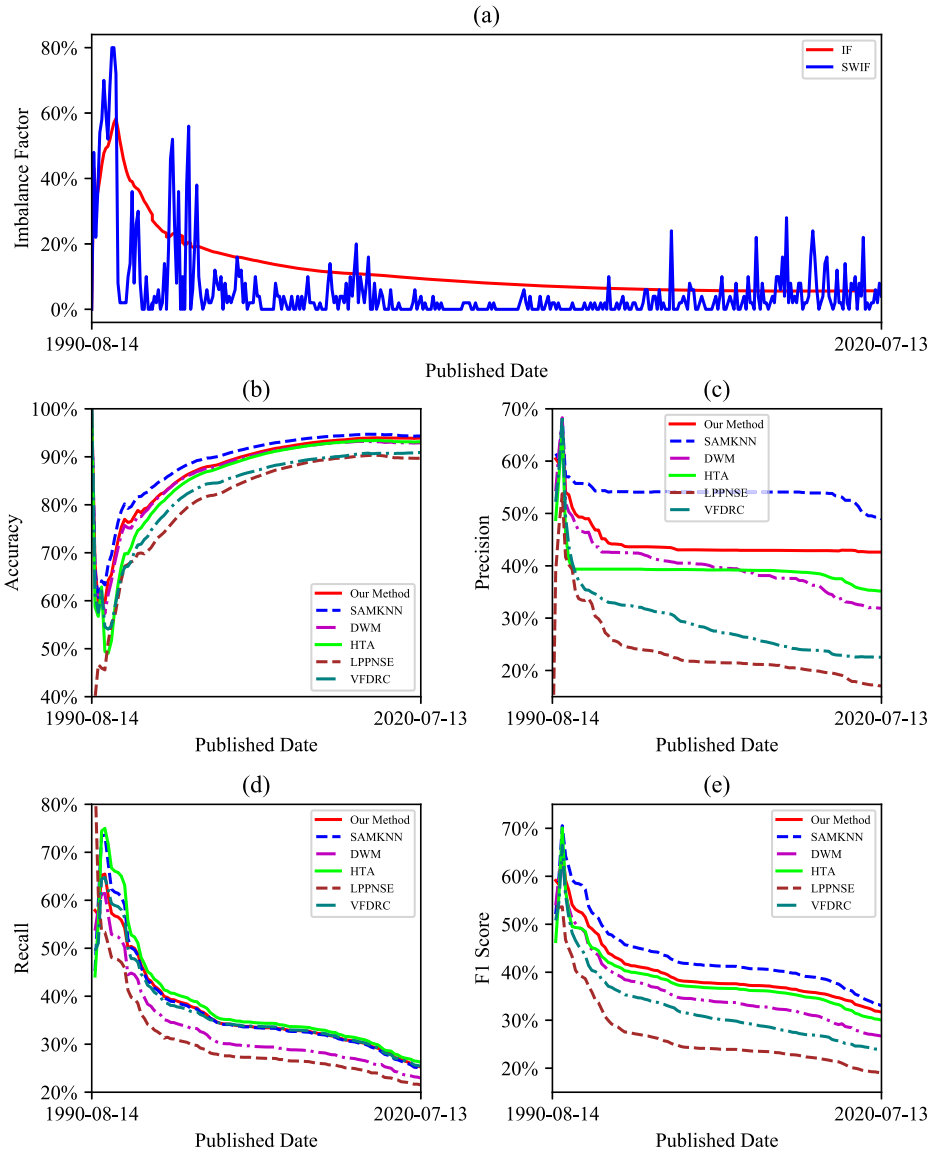

Figure 6 Realtime classification results on class 'ZeroDay'

Table 7 gives the overall classification result on all existing samples of class 'ZeroDay'. SAMKNN also performs the best on Accuaray, Precision and F1 Score. HTA achieves the best Recall. our method is the runner-up on all metrics. HTA is also in the third place of

Table 7 Overall classification results on all existing samples of class 'ZeroDay'

Algorithms	Accuracy	Precision	Recall	F1 Score
Our Method	93.89%	42.62%	25.25%	31.71%
SAMKNN	94.32%	48.96%	24.94%	33.05%
DWM	92.91%	31.89%	22.97%	26.70%
HTA	93.13%	35.13%	26.24%	30.04%
LPPNSE	89.71%	17.05%	21.52%	19.03%
VFDRC	90.90%	22.54%	25.40%	23.88%

the overall performance measured by F1 Score. The worst F1 Score belongs to LPPNSE with 19.03%, which is also quite good, considering that class 'ZeroDay' only accounts for 5.62%.

4.4.3 Classification result on the minority classes—'Pos'

Finally, we present the realtime classification result on class 'pos' in Figure 7. HTA, SAMKNN and our methods are at the first level in regard to Accuracy and Precision. However, both HTA and SAMKNN have very poor performance on Recall and F1 Score. our method, along with LPPNSE and VFDRC, achieves the best F1 Score most of the times.

Table 8 displays the overall classification result on all existing samples of class 'Pos'. HTA achieves the best Accuracy and Precision and DWM performs best on Recall. Taking F1 Score as the overall performance of an algorithm, our method obtains the best at 49.97%, 5.69% higher than the second one, 45.52% achieved by DWM.

4.4.4 Average classification results on multiple classes

As discussed above, not a single algorithm achieves the best on all classes or all metrics. We further discuss the performance of the afore-mentioned algorithms by calculating their Macro and Micro average classification performance. As shown in Table 9, our method achieves the best Macro F1 Score with 56.62%, which is 1.36% higher than the second place, SAMKNN, and 3.73% higher than the third place (HTA).

With respect to the Micro F1 Score, SAMKNN achieves the best with 79.85, which is only 0.19% better than our method and 0.70% higher than HTA, the third best algorithm.

The results show that our method is among the best of all these six algorithms. Besides, our method is also the most robust algorithm, which can always achieve the best or the equivalent best performance on different metrics and classes.

5 Conclusions

Vulnerability exploitation time prediction is of importance for vulnerability assessment and cybersecurity management. In order to provide results with finer granularity and adopt

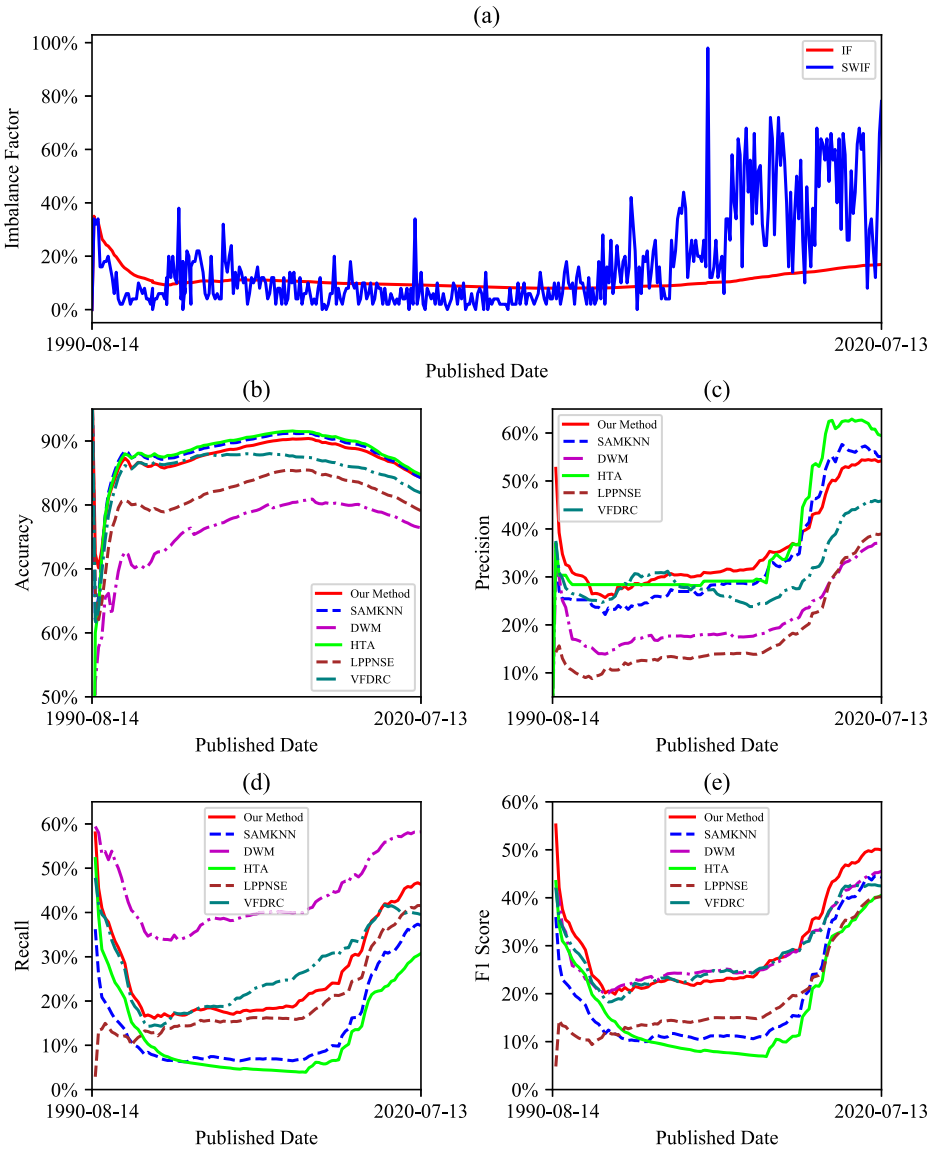

Figure 7 Realtime classification results on class 'pos'

the real-world online learning situation, we treat this task as a data stream classification problem.

We propose an integrated consecutive batch learning framework to predict the exploitation time of vulnerabilities. We further propose an ASWWL algorithm to handle the existing

Table 8 Overall classification results on all existing samples of class 'Pos'

Algorithms	Accuracy	Precision	Recall	F1 Score
Our Method	84.29%	54.19%	46.35%	49.97%
SAMKNN	84.24%	55.07%	37.03%	44.28%
DWM	76.42%	37.37%	58.22%	45.52%
HTA	84.74%	59.45%	30.74%	40.53%
LPPNSE	79.09%	38.96%	41.58%	40.22%
VFDRC	81.87%	45.84%	39.53%	42.45%

dynamic multiclass imbalance problem in data stream scenario. Experiments conducted on real-world vulnerabilities collected between 1990 and 2020 show that the ASWWL algorithm is effective on booting the classification performance of the minority classes without compromising the performance of the majority class. We also compare our method with other five data stream leaning algorithms. Results show that our method is the most robust algorithm on different metrics and classes. The performance of our method is also among the best of all these six algorithms.

Our research is a practice in vulnerability assessment and cybersecurity management. Although performance improvement has been achieved in vulnerability exploitation time prediction, further work still needs to be done for vulnerability assessment and cybersecurity management. In future, we will explore the exploitation time prediction with much finer granularity to lead to a better understanding of the risks of vulnerabilities.

Table 9 Average classification results on multiple classes

Average	Algorithms	Accuracy	Precision	Recall	F1 Score
Macro	Our Method	79.66%	60.82%	54.16%	56.62%
	SAMKNN	79.85%	62.73%	51.72%	55.26%
	DWM	72.07%	52.27%	53.28%	51.70%
	HTA	79.15%	59.28%	50.18%	52.89%
	LPPNSE	71.50%	46.84%	48.25%	47.44%
	VFDRC	74.53%	50.79%	50.22%	50.39%
Micro	Our Method	79.66%	79.66%	79.66%	79.66%
	SAMKNN	79.85%	79.85%	79.85%	79.85%
	DWM	72.07%	72.07%	72.07%	72.07%
	HTA	79.15%	79.15%	79.15%	79.15%
	LPPNSE	71.50%	71.50%	71.50%	71.50%
	VFDRC	74.53%	74.53%	74.53%	74.53%

Acknowledgements The first author was partially supported by the Research Program of Chongqing University of Arts and Sciences, China (Grant No. P2020RG08) and the Natural Science Foundation of Chongqing, China (Grant No. cstc2019jcyj-msxmX034).

References

1. Afzaliseresht, N., Miao, Y., Michalska, S., Liu, Q., Wang, H.: From logs to stories: human-centred data mining for cyber threat intelligence. IEEE Access **8**, 19089–19099 (2020)
2. Alazab, M., Tang, M.: Deep Learning Applications for Cyber Security. Springer, Berlin (2019)
3. Anwar, M.M., Liu, C., Li, J.: Discovering and tracking query oriented active online social groups in dynamic information network. World Wide Web **22**(4), 1819–1854 (2019)
4. Bifet, A., Gavaldà, R.: Adaptive learning from evolving data streams. In: International Symposium on Intelligent Data Analysis, pp. 249–260. Springer (2009)
5. Bozorgi, M., Saul, L.K., Savage, S., Voelker, G.M.: Beyond heuristics: learning to classify vulnerabilities and predict exploits. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 105–114. ACM (2010)
6. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805 (2018)
7. Du, J., Michalska, S., Subramani, S., Wang, H., Zhang, Y.: Neural attention with character embeddings for hay fever detection from twitter. Health Inf. Sci. Sys. **7**(1), 1–7 (2019)
8. Edkrantz, M., Said, A.: Predicting cyber vulnerability exploits with machine learning. In: SCAI, pp. 48–57 (2015)
9. Elwell, R., Polikar, R.: Incremental learning of concept drift in nonstationary environments. IEEE Trans. Neural Netw. **22**(10), 1517–1531 (2011)
10. Hulten, G., Spencer, L., Domingos, P.: Mining time-changing data streams. In: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 97–106 (2001)
11. Islam, M.R., Kabir, M.A., Ahmed, A., Kamal, A.R.M., Wang, H., Ulhaq, A.: Depression detection from social network data using machine learning techniques. Health Inf. Sci. Sys. **6**(1), 1–12 (2018)
12. Jacobs, J., Romanosky, S., Adjerid, I., Baker, W.: Improving vulnerability remediation through better exploit prediction. J. Cybersec. **6**(1), tyaa015 (2020)
13. Jacobs, J., Romanosky, S., Edwards, B., Roytman, M., Adjerid, I.: Exploit prediction scoring system (epss). arXiv:1908.04856 (2019)
14. Kolter, J.Z., Maloof, M.A.: Dynamic weighted majority: an ensemble method for drifting concepts. J. Mach. Learn. Res. **8**, 2755–2790 (2007)
15. Kosina, P., Gama, J.: Very fast decision rules for classification in data streams. Data Min. Knowl. Disc. **29**(1), 168–202 (2015)
16. Li, H., Wang, Y., Wang, H., Zhou, B.: Multi-window based ensemble learning for classification of imbalanced streaming data. World Wide Web **20**(6), 1507–1525 (2017)
17. Li, M., Sun, X., Wang, H., Zhang, Y., Zhang, J.: Privacy-aware access control with trust management in web service. World Wide Web **14**(4), 407–430 (2011)
18. Li, Z., Wang, X., Li, J., Zhang, Q.: Deep attributed network representation learning of complex coupling and interaction. Knowl.-Based Syst. **212**, 106618 (2021)
19. Losing, V., Hammer, B., Wersing, H.: Knn classifier with self adjusting memory for heterogeneous concept drift. In: 2016 IEEE 16th International Conference on Data Mining (ICDM), pp. 291–300 (2016)
20. Montiel, J., Read, J., Bifet, A., Abdessalem, T.: Scikit-multiflow: a multi-output streaming framework. J. Mach. Learn. Res. **19**(72), 1–5 (2018). http://jmlr.org/papers/v19/18-251.html
21. Rasool, R.U., Ashraf, U., Ahmed, K., Wang, H., Rafique, W., Anwar, Z.: Cyberpulse: a machine learning based link flooding attack mitigation system for software defined networks. IEEE Access **7**, 34885–34899 (2019)
22. Sarki, R., Ahmed, K., Wang, H., Zhang, Y.: Automated detection of mild and multi-class diabetic eye diseases using deep learning. Health Inf. Sci. Sys. **8**(1), 1–9 (2020)
23. Shen, Y., Zhang, T., Wang, Y., Wang, H., Jiang, X.: Microthings: a generic iot architecture for flexible data aggregation and scalable service cooperation. IEEE Commun. Mag. **55**(9), 86–93 (2017)
24. Tang, M., Yin, J., Alazab, M., Cao, J.C., Luo, Y.: Modelling of extreme vulnerability disclosure in smart city industrial environments. IEEE Trans. Indust. Inf., pp. 1–1 (2020)

25. Tang, M., Alazab, M., Luo, Y.: Big data for cybersecurity: vulnerability disclosure trends and dependencies. IEEE Trans. Big Data **5**(3), 317–329 (2019)
26. Tavabi, N., Goyal, P., Almukaynizi, M., Shakarian, P., Lerman, K.: Darkembed: exploit prediction with neural language models. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
27. Vimalachandran, P., Liu, H., Lin, Y., Ji, K., Wang, H., Zhang, Y.: Improving accessibility of the australian my health records while preserving privacy and security of the system. Health Inf. Sci. Sys. **8**(1), 1–9 (2020)
28. Wang, H., Sun, L., Bertino, E.: Building access control policy model for privacy preserving and testing policy conflicting problems. J. Comput. Syst. Sci. **80**(8), 1493–1503 (2014)
29. Wang, H., Wang, Y., Taleb, T., Jiang, X.: Special issue on security and privacy in network computing. World Wide Web **23**(2), 951–957 (2020)
30. Wang, H., Yi, X., Bertino, E., Sun, L.: Protecting outsourced data in cloud computing through access management. Concur. Comput. Pract. Exp. **28**(3), 600–615 (2016)
31. Wang, S., Minku, L.L., Yao, X.: A learning framework for online class imbalance learning. In: 2013 IEEE Symposium on Computational Intelligence and Ensemble Learning (CIEL), pp. 36–45 (2013)
32. Wang, S., Minku, L.L., Yao, X.: Dealing with multiple classes in online class imbalance learning. In: IJCAI, pp. 2118–2124 (2016)
33. Yi, X., Zhang, Y.: Privacy-preserving distributed association rule mining via semi-trusted mixer. Data Knowl Eng **63**(2), 550–567 (2007)
34. Yin, J., Cao, J., Siuly, S., Wang, H.: An integrated mci detection framework based on spectral-temporal analysis. Int. J. Autom. Comput. **16**(6), 786–799 (2019)
35. Yin, J., Tang, M., Cao, J., Wang, H.: Apply transfer learning to cybersecurity: predicting exploitability of vulnerabilities by description. Knowl-Based Sys., pp. 106529. https://doi.org/10.1016/j.knosys.2020.106529 (2020)
36. Yin, J., Tang, M., Cao, J., Wang, H., You, M., Lin, Y.: Adaptive online learning for vulnerability exploitation time prediction. In: Web Information Systems Engineering – WISE 2020, pp. 252–266. Springer (2020)
37. Yin, J., You, M., Cao, J., Wang, H., Tang, M., Ge, Y.F.: Data-driven hierarchical neural network modeling for high-pressure feedwater heater group. In: Australasian Database Conference, pp. 225–233. Springer (2020)
38. Zhang, F., Wang, Y., Liu, S., Wang, H.: Decision-based evasion attacks on tree ensemble classifiers. World Wide Web **23**(5), 2957–2977 (2020)
39. Zhang, J., Li, H., Liu, X., Luo, Y., Chen, F., Wang, H., Chang, L.: On efficient and robust anonymization for privacy protection on massive streaming categorical information. IEEE Trans Depend Sec Comput **14**(5), 507–520 (2015)
40. Zhang, J., Tao, X., Wang, H.: Outlier detection from large distributed databases. World Wide Web **17**(4), 539–568 (2014)