Neurocomputing 520 (2023) 115–128

Contents lists available at ScienceDirect

Neurocomputing

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

Reinforcement learning based web crawler detection for diversity and dynamics

Yang Gao [a], Zunlei Feng [b,*], Xiaoyang Wang [a], Mingli Song [a], Xingen Wang [a], Xinyu Wang [a], Chun Chen [a]

[a] College of Computer Science, Zhejiang university, Hangzhou, China
[b] College of Software Technology, Zhejiang University, Hangzhou, China

ARTICLE INFO

Article history:
Received 9 July 2022
Revised 26 October 2022
Accepted 15 November 2022
Available online 19 November 2022
Communicated by Zidong Wang

Keywords:
Web crawler detection
Reinforcement learning
Feature selection
Crawler diversity
Crawler dynamics

ABSTRACT

Crawler detection is always an important research topic in network security. With the development of web technology, crawlers are constantly updating and changing, and their types are becoming diverse. The diversity and dynamics of crawlers pose significant challenges for feature applicability and model robustness. Existing crawler detection methods can only detect a limited number of crawlers by predefined rules and can not cover all types of crawlers; worse, they can be completely invalidated by the emergence of new types of crawlers. In this paper, we propose a reinforcement learning based web crawler detection method for diversity and dynamics (WC3D), which is composed of a feature selector and a session classifier. The feature selector selects the appropriate feature set for different types of crawlers with deep deterministic policy gradient. The session classifier makes crawler detection and provides rewards to the feature selector. The two modules are trained jointly to optimize the feature selection and session classification processes. Extensive experiments demonstrate the existence of crawler diversity and that the proposed method is still highly robust against the new type of crawlers and achieves state-of-the-art performance even without considering the dynamics of the crawlers.

1. Introduction

Web crawlers (also known as web spiders, web robots) are programs or scripts that automatically crawl web information with certain rules [1]. According to statistics, more than half of today's network requests are made by crawlers [2,3]. These crawlers include many malicious crawlers, which ignore the constraints of robots.txt [4], infringe on user privacy, endanger network security, and cause network traffic overload. What they do seriously affects the user's online experience [5–7]. Therefore, how to detect crawlers in numerous network requests has become an important research topic in the network security field.

With the continuous development of Web technology, the enrichment of Web content, and the advancement of crawler detection research, crawlers are constantly being updated and changed. The types of crawlers have become diverse, including search engine crawlers that collect web content, topic crawlers that crawl relevant content according to a specified topic, and image crawlers that are only interested in images [8,9]. Crawlers' diversity and dynamics bring the following challenges to crawler detection:

1. It isn't easy to extract universal and stable features. Different types of crawlers have some differences in the distribution of feature space. Some features that apply to some crawlers' detection do not apply to other types of crawlers. For example, image crawlers are highly differentiated from users for the feature of Image-request-ratio, while non-image crawlers are consistent with users. Crawler detection requires valid features for different types of crawlers;
2. New types of crawlers make pre-trained models or rules invalid. Some new types of crawlers that are not distinguishable in the mode's features will decrease the detection accuracy, and the model is no longer robust.

Many scholars start from the perspective of features, hoping to design universal and stable features to characterize crawlers, such as Resource-request-patterns, which characterizes the pattern of web request file types [10]; Penalty, which characterizes the behavior of back-and-forward navigation or loop, and Maximum-rate-of-browser-file, which characterizes the properties of the session for resource requests, etc [11]. Such methods detect crawlers by using

* Corresponding author.
E-mail addresses: roygao@zju.edu.cn (Y. Gao), zunleifeng@zju.edu.cn (Z. Feng), skyoung@zju.edu.cn (X. Wang), brooksong@zju.edu.cn (M. Song), newroot@zju.edu.cn (X. Wang), wangxinyu@zju.edu.cn (X. Wang), chenc@zju.edu.cn (C. Chen).

https://doi.org/10.1016/j.neucom.2022.11.059
0925-2312/

artificially defined feature sets, as shown in the first row of Fig. 1. As the number of features increases, some scholars have added feature selection methods, hoping to remove redundant features and keep the robust ones [12,13]. Such methods use feature selection to determine the feature set used for crawler detection, as shown in the second row of Fig. 1. These methods solve the problem of crawlers' diversity to a certain extent, but they also have some disadvantages:

1. the feature set is single, and all types of crawlers use the same feature set, and some features that can strongly characterize a specific type of crawlers are not selected;
2. the feature selection process is separated from the crawler detection task, and most of the selection methods using a greedy strategy, which easily causes the local optimal problem.

The last and the essential point is that the feature set used can not change dynamically. As the crawlers update and change, the selected features may become invalid, while the previously unselected features may become effective. And the robustness of the model will be significantly reduced when the used features are no longer suitable for distinguishing the new type of crawlers.

To address the challenges posed by the diversity and dynamics of crawlers, we want to propose a feature selection and detection method for the new types of crawlers that can sense the diversity of crawlers and dynamically select their characterization feature sets according to their types. We introduce a reinforcement learning method in which we model the feature selection as a sequential decision process, and we use the classification accuracy as the reward to optimize the screening process, which can avoid the local optimal problem; at the same time, we use the feature space of the crawler as the state space, and the agent not only learns the potential connection between features but also perceives more unknown feature distributions through exploration, which allows the model to better adapt to the updates and changes of crawlers and the emergence of the new type of crawlers. Our approach consists of two modules, a feature selection module based on deep reinforcement learning and a classification module composed of deep neural networks. The feature selection module optimizes the filtering process based on the results of the classification module, and the classification module uses the feature set provided by the feature selection module to detect crawlers. As shown in the third row of Fig. 1, our method can select different feature sets for different types of crawlers to better accommodate the diversity of crawlers and still achieve high robustness in detecting the new types of crawlers.

The contributions of this paper are as follows:

1. We propose a feature selection method for the diversity of crawlers, and the selected feature set has diversity and higher applicability.
2. We propose a crawlers detection method based on reinforcement learning, which can maintain high detection accuracy when crawlers have diversity and dynamics and have good robustness in the face of the new types of crawlers.
3. There are few public data sets in the field of crawler detection, which has caused certain obstacles to the development of research in this field. This paper publicizes a crawler data set for scholars' follow-up research convenience.

The content of this paper is organized as follows: In the second part, we introduce the recent work in the field of crawler detection. The third part introduces the feature selection and classification method for the new types of crawlers based on deep reinforcement learning. We give the experimental results in the fourth part. The fifth part is the summary of this article.

2. Related work

In this part, we will review some recent research in crawler detection, crawler diversity and feature selection.

A web crawler is a program or script that automatically crawls the World Wide Web for information according to specific rules. Well-intentioned crawlers have the functions of communicating with users, maintaining mirror sites, testing the validity of web pages, etc. However, with the development of web technology, more and more malicious crawlers have emerged, extorting users, illegally collecting information, DDos attacks, stealing users' privacy, and taking up network bandwidth to affect users' online experience. Due to the diversity of websites and the diversity of crawling rules, crawlers also have significant differences. Some of them crawl only specific resources while others collect all types of resources; some use breadth-first strategy to crawl content while others use depth-first; in addition, with the advancement of research on crawler detection, crawlers are constantly updated and changed to better mimic user behavior and thus evade website detection. The crawlers' diversity and dynamics pose a more significant challenge to the accuracy of crawler detection. Scholars have done a lot of research in this area, and they classify crawler detection methods into the following four categories [14]: syntactic log analysis, traffic pattern analysis, Turing test systems and analytical learning techniques. Syntactic log analysis: This method identifies crawlers based on log files by matching keywords in logs or matching IP and user agents with the whitelist. This method is relatively straightforward. Some initial research for crawler detection is based on this method [15,16]. Traffic pattern analysis: This method analyzes the differences in access patterns between crawlers and users and detects crawlers by comparing the access patterns obtained by simulating a specified algorithm (such as DFS or BFS) to crawlers and calculating the similarity between them [17–21]. Turing test systems: This method uses the Turing test for real-

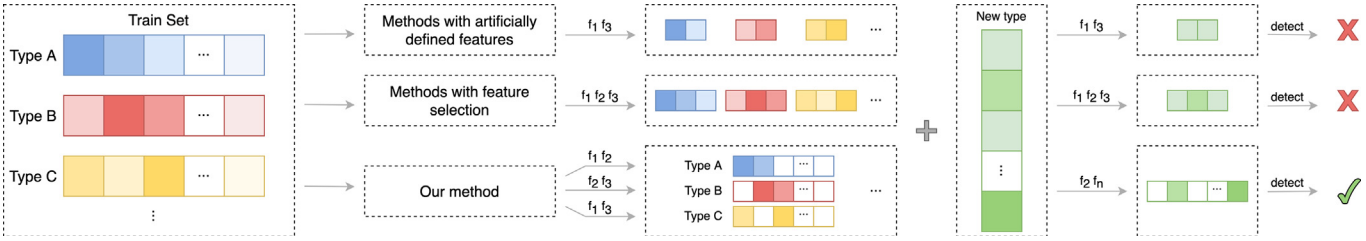


Fig. 1. Illustrative the challenges posed by the diversity and dynamics of crawlers. The diversity of crawlers makes the distribution of feature space different. Each type has its appropriate feature set for representing its characteristics, and the darker the color of the features in the graph, the more suitable for representing. The first row indicates the method with an artificially defined feature set, which uses the first and third features for crawler detection, the second row indicates the method with feature selection approach, which retains the first three features after filtering, and the third line indicates that our approach selects theirs appropriate feature set according to the different types of crawlers. When the new type of crawlers appear, the features used in the first two methods can not characterize them well and will lead to a decrease in detection accuracy.

time detection of crawlers. This method requires software-level modifications to the website. A more intuitive understanding is that the website distinguishes crawlers from users through verification codes. Some studies have adopted this method [22,23]. Analytical learning techniques: This method extracts features from sessions and uses machine learning to obtain detection results. With the continuous development of machine learning, many scholars have adopted different models and features for crawler detection. They have obtained relatively ideal results, which proves the effectiveness of this type of method. Our crawler detection model based on reinforcement learning also belongs to the category of analytical learning techniques, so we will focus on the related work of this method in the following text.

Some scholars hope to solve crawler diversity and dynamics by proposing stable and universal features for different types of crawlers. Tan, Kumar et al. were the first to propose 25 new features and use C4.5 decision trees to classify website visitors, which contain some statistical features such as the proportion of nightly visits, unassigned referrer, session time, etc. These statistical features were also frequently used in subsequent studies [1]. Doran, Gokhale et al. proposed a resource request pattern feature that represents the sequence of requests for different types of files, and the authors used a probabilistic model to analyze the differences between crawlers and users on this feature [10]. Lagopoulos et al. propose a crawler detection method for Linear Discriminant Analysis(LDA) based on the assumption that users are only interested in specific topics and crawlers will randomly crawl various topics. The authors also propose some novel features that can capture the semantics of the requested resource content, which improve the crawler detection accuracy [24]. Zabihi, Jahan et al. proposed two new features in their paper: Maximum rate of browser file request and penalty. The former characterizes the number property of all resource files bound to the visited page. The latter characterizes the behavior of web request back-and-forward navigation or loop. The authors concluded that these two features do not change over time, and they used the Density-Based Spatial Clustering of Applications with Noise(DBSCAN) algorithm for crawler detection [12]. Hiltunen et al. used the number and sequence of visited pages as features and used the Self-organizing Maps(SOM) method for crawler detection [25]. Zhu, Gao et al. proposed a machine learning-based approach that combines real-time monitoring with offline detection with two new heuristic rules. Eventually, they used nine features for crawler detection, and the model was able to achieve both high recognition accuracy and fast response time [26]. These methods use human-defined feature sets and combine different machine learning methods for crawler detection, which address the challenges to the applicability of features posed by the diversity of crawlers to some extent.

As the number of features increases, there are more and more redundant features. Some scholars believe that these redundant features increase the complexity of the model [27–29] and the feature set defined by humans has a solid subjective nature, which does not objectively judge the merits of the features, so they use feature selection to decide the feature set used for crawler detection. Many scholars use decision trees for feature filtering, Kwon et al. used decision trees to classify web sessions and proposed some new behavioral features of web crawlers based on features from previous studies [18]. Toni et al. proposed an intelligent system called Lino to identify whether a web crawler is malicious or not. They used support vector machine and C4.5 decision tree algorithm respectively, and compared the performance of these two algorithms in the Lino system [30]. Hamidzadeh et al. pointed out that website visitors have variability, crawlers' tasks have diversity, and especially the content of websites also has diversity. These three problems indicate the need for a crawler detection method that can dynamically select features according to the content of different websites for different tasks. They used the Fuzzy Rough Set(FRS) algorithm to select the features and classified them using the SOM method. The FRS algorithm was able to consider the similarity of all features simultaneously and deal with the ambiguity of the data effectively. Finally, it also achieved better experimental results [13]. Zabihi et al. used t-test to select the features with higher relevance among the 14 valid attributes. Although their proposed method has effective performance in distinguishing between crawlers and human users, the t-test is only based on the summary statistics of the compared attributes and has a more limited observation of the data [31]. The above study adds feature selection process to crawler detection, which reduces the complexity of the model by sifting out redundant features and makes the selected features more interpretable while solving the feature applicability problem.

The two types of methods mentioned above both start from the feature perspective and propose or select stable and universal features to address the challenges of the diversity of crawlers. Although these methods have achieved specific results, they still have some defects: 1. The feature set used is single; all types of crawlers use the same feature set, and some features that can well characterize a specific type of crawlers are not selected; 2. Some feature selection methods separated from the detection task tend to make the model fall into the local optimum problem. 3. Most importantly, they do not address the problem of dynamics of crawlers. As crawlers update and change, the feature set they use may not be suitable for the new types of crawlers, and the previously unselected features may become more effective. Doran, Gokhale et al. point out that crawler behavior patterns may change over time and that the features proposed by scholars are a weak reflection of crawler and user behavior, which may not reflect the essential differences between them [10].

To address these issues, we need a method that can perceive the diversity of crawlers and adjust the representation of crawler behavior patterns according to the dynamics of crawlers. In the problem of perceiving dynamic changes, reinforcement learning has always performed well [32,33], and in recent years some scholars have applied reinforcement learning to the problem of feature selection. Janisch et al. use Q-learning for feature selection and classification, and they design a reinforcement learning model with two types of actions in the action space. One is to add the unselected features to the feature set, and the other is to classify directly with the current set of selected features. They combined the classification task with the feature selection task, and the results of classification affect the feature selection process as a reward [34]. Xu, Wang et al. used reinforcement learning to extract the more important features from the temporal features, and the importance of their features was also influenced by the classification results [35]. Feng, Huang et al. also used reinforcement learning to improve the accuracy of the classification task; instead of using reinforcement learning to filter features, they filtered data used to train the model, but the principle is the same [36]. Inspired by the above studies, we wish to apply reinforcement learning to the crawler detection problem to solve the problem of sensing crawler diversity and dynamically adjusting the crawler's representational patterns. To this end, we propose a reinforcement learning-based crawler feature selection and detection method, which will be described in detail in the next section.

3. Method

To solve the problem of crawlers' diversity and dynamics mentioned above, we propose a new crawler detection method based

on reinforcement learning, which can sense the diversity of crawlers and dynamically select their appropriate set of representative features for different types of crawlers. Our method avoids the problems of single and static feature sets, making it suitable for detecting the new type of crawlers.

Our model consists of two modules: the feature selector module, which is responsible for selecting a feature set suitable for characterizing the session according to its feature distribution, and a session classifier module, which is responsible for classifying a session using the feature set provided by the feature selector. These two modules interact during the training process to achieve the best detection results.

3.1. Problem definition

Formally, we decompose the task of web crawler detection with diversity and dynamics into two sub-problems in this paper: feature selection and session classification.

We formulate the feature selection problem as follows: given the known multi-type crawlers set as $C = \{c_1, c_2, \ldots, c_m\}$, where c_i is a type of crawler that we have observed and an unknown new type of crawler that we have not observed as $c\prime$. Different types of crawlers have different characteristics and feature distribution. The goal of feature selection is to determine which features can better represent its characteristic for different crawlers according to the known crawlers set C, and it still works when dealing with unknown crawler $c\prime$. The output of feature selection is the feature masks as $M = \{m_1, m_2, \ldots, m_m, m\prime\}$, where m_i is the feature mask of ith type of crawler that we have observed and $m\prime$ is the feature mask of the new type of crawler, and $m_i, m\prime = \{k_1, k_2, \ldots, k_n\}$, where k_i is the ith feature's mask and is 1 or 0, which means selecting or discarding this feature.

The session classification problem is formulated as follows: given the initial feature vector of a session as $s = \{f_1, f_2, \ldots f_n\}$ and its feature mask as $m = \{k_1, k_2, \ldots k_n\}$, we can get the final feature vector as $s\prime = \{f\prime_1, f\prime_2, \ldots, f\prime_n\}$ where $f\prime_i$ is the result of multiplying the ith feature f_i with its feature mask k_i, the goal is to predict the likelihood that it will be a crawler or user under this final feature vector.

3.2. Overview

The proposed model consists of the feature selector and the session classifier. The feature selector is based on deep deterministic policy gradient with sessions as input. Each session s_i belongs to a specific type of crawler or user and has a corresponding action a_i to indicate its feature mask. Different types of crawlers can obtain different feature masks. And the feature mask represents which features should be retained or discarded to represent its characteristics better. The session classifier adopts a deep neural network. It uses the initial feature vector and feature mask from the feature selector as input to classify sessions. Meanwhile, the session classifier gives feedback to the feature selector to refine its policy. Fig. 2 gives an illustration of how the proposed model works.

With the help of the feature selector based on reinforcement learning, our method can select the most suitable feature set for different types of crawlers, avoiding the problem that features screening results are simplified and static. And using reinforcement learning can avoid the problem of local optimization. All of our designs aim at making the model work well in complex network environments with crawlers' diversity and dynamics.

3.3. Feature selector

We cast feature selector as a reinforcement learning problem. As we all know, reinforcement learning needs an agent and environment. In our method, the agent is the feature selector, and the environment is the session classifier. The agent follows a policy to decide which features should be retained and output feature mask and then receive a reward depending on the accuracy of classification from the session classifier. We will introduce the state, action, state transition, terminal state, and reward as follows.

- **State**
  The state s_i represents the current feature distribution of a session, which can be formulated as: $s_i = \{f_1, f_2, \ldots, f_n\}$, where f_i is a specific value of a feature, and n is the number of features. The initial state of a session is its feature distribution in the dataset, which is extracted from a log file. With the screening of the feature selector, its feature distribution (state) will also change.
- **Action**
  First, we define the output of policy network as $p_i = \{w_1, w_2, \ldots, w_n\}$ to indicate the importance of each feature, where $w_i \in (0, 1)$, corresponding to the weight value assigned by the agent to the ith feature of the session, and a larger value of w_i indicates that the ith feature is more critical to this session. Then we get the action a_i by the following formula:
  $$a_i = T(\mu(s_i|\theta^{\mu}) + \mathcal{N}_i) \tag{1}$$
  where μ is the policy network, $\mathcal{N}_i$ is Uhlenbeck-ornstein random process [37], θ^{μ} is the parameters of policy network and T is a mapping function that uses the predefined threshold t_1 to map the weights w_i to feature mask m_i. If w_i is greater than or equal to t_1, then the corresponding m_i is 1, otherwise 0. Thus the action can be formulated as $a_i = \{m_1, m_2, \ldots, m_n\}$, and $m_i \in \{0, 1\}$, which means discarding or retaining the ith feature.
- **State Transition**
  We define the state transition as $s_t \times a_t \rightarrow s_{t+1}$, which means we multiply the current feature distribution by its mask to obtain a new feature distribution (the next state).
- **Terminal State**
  The definition of the terminal state is required in the modeling of reinforcement learning, and no more state transfer will be performed when the agent reaches the terminal state. Our task is to classify the session correctly, so we use the logarithm of the output probability of the session classifier as a condition to determine whether the agent enters the terminal state. When $logP(c_g|s_i)$ is greater than our predefined threshold t_2, e.g., 0.8, we consider that the current feature distribution is sufficient to characterize the session, and the agent enters the terminal state, then its feature distribution will no longer change.
- **Reward**
  We use the logarithm of the output probability of session classifier to calculate the reward: $P(y = c_g|s_i)$, where c_g is the gold label of the input session s_i. In addition, to encourage the model to screen out more features that can not characterize the session, we include an additional term by computing the proportion of the number of discarded features to the number of all features. When the agent enters the terminal state, we use the classification result as the reward, giving a larger reward if the classification is correct and a negative reward if the result is wrong. The reward can be described as below:
  $$r_i = \begin{cases} 1 & \text{if } s_i \in \mathcal{T}, y = c_g \\ logP(c_g|s_i) + \alpha N'/N & \text{if } s_i \notin \mathcal{T} \\ -1 & \text{if } s_i \in \mathcal{T}, y \neq c_g \end{cases}$$

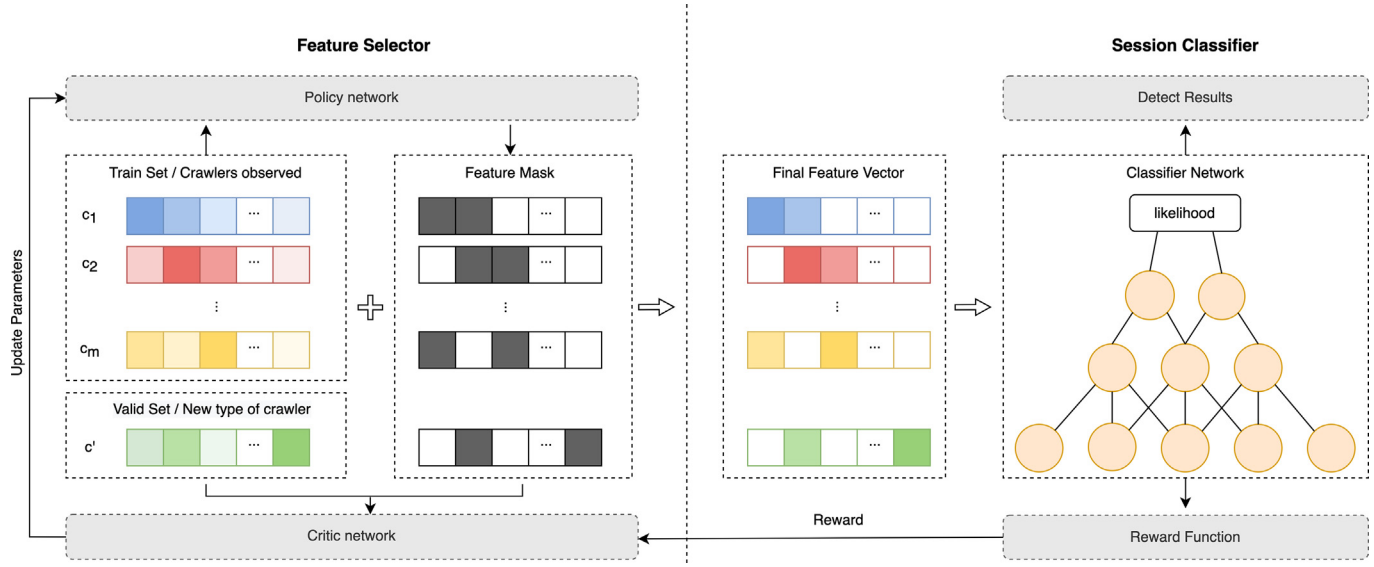


Fig. 2. Overview of model. The training set contains several types of crawlers, and these crawlers will get different feature masks after the policy network and then get different feature sets. The processed feature vectors are input to the session classifier to get the detection results. The reward function calculates the reward feedback to the critic network and then optimizes the policy network to produce a better feature set. After training, the new type of crawler, which is not included in the training set, is used to detect the robustness of the model.

where $\mathscr{T}$ is the terminal state, $N\prime$ denotes the number of discarded features, N denotes the number of all features, and α is a hyper-parameter to balance the two terms.

Based on the above definition of reinforcement learning model elements, our method uses the architecture of Deep Deterministic Policy Gradient(DDPG), which is an actor-critic based algorithm in reinforcement learning [37]. DDPG use a policy network to generate the actions of the agent. The Critic network can judge the quality of actions and guide the updating direction of policy function. We will introduce the policy network and the critic network in the following.

- **Policy Network**
  In our method, the policy network consists of several fully connected layers and the policy function is formulated as follows:

$$\mu(s_i|\theta^\mu) = \sigma(W_\mu * s_i + b_\mu) \tag{2}$$

where $\sigma(\cdot)$ is sigmoid function with the parameter of policy network$\theta^\mu = \{W_\mu, b_\mu\}$. The policy network takes the feature distribution of the session (state) as input and outputs the feature mask (action) corresponding to the input. We use the sigmoid function to deflate the mask to (0, 1), representing the weights assigned to each feature. To balance the exploration and exploitation problem in reinforcement learning, we use the Uhlenbeck-Ornstein stochastic process as the random noise.
- **Critic Network**
  DDPG uses a neural network to model the Q function like Deep Q Network (DQN), so the Q function can be formulated as follows:

$$Q(s_i, a_i|\theta^Q) = W_Q * F(s_i, a_i) + b_Q \tag{3}$$

where $F(s_i, a_i)$ is a vector that combines state s_i and action a_i with the parameter of critic network $\theta^Q = \{W_Q, b_Q\}$. The critic network takes the combined vector of states and actions as input and outputs the Q value of the action corresponding to the current state.

- **Optimization**
  The previous practice has shown that if only a single Q neural network algorithm is used, the learning process is volatile because the parameters of the Q network are used to calculate the gradient of the critic network and the policy network, while the gradient update is frequent, based on this, DDPG creates two copies of neural networks for the policy network and critic network respectively, called target network. Thus, our feature selector consists of 4 networks: policy network, critic network, target policy network, and target critic network.
  For the policy network, we use a function J to measure the performance of the policy $\mu(s_i|\theta^\mu)$, which we call the performance objective, defined as follows:

$$J_\beta(\mu) = \int_S \rho^\beta(s) Q^\mu(s, \mu(s)) ds \tag{4}$$

where s is the state which has the distribution function $\rho^\beta, Q^\mu(s, \mu(s))$ is the Q value that generated in each state if acton $\mu(s)$ is selected in accordance with the policy $\mu(s_i|\theta^\mu)$. The goal of our training is to maximize $J_\beta(\mu)$. According to the mathematical derivation of the Deterministic Policy Gradient(DPG) [38], the policy gradient can be formulated as follows:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \cdot \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s=s_i} \tag{5}$$

where N is the number of mini-batch data, and this is an unbiased estimate of policy gradient with the method of Monte-Carlo.
For the critic network, in order to get a more accurate Q value, we use a supervised learning-like approach to calculate its gradient with Mean Square Error(MSE) loss:

$$L_Q = \frac{1}{N}\sum_i (r_i + \gamma Q\prime(s\prime_i, \mu\prime(s\prime_i|\theta^{\mu\prime})|\theta^{Q\prime}) - Q(s_i, a_i|\theta^Q))^2 \tag{6}$$

where r_i is the reward, $Q\prime$ is the target critic network, $\mu\prime$ is the target policy network, $s\prime_i$ is the next state of s_i and $\gamma \in [0, 1]$ is discounted rate.
For the update of the target policy network and target critic network, we use a soft update algorithm as follows:

$$\theta^{\mu\prime} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu\prime} \tag{7}$$

$$\theta^{Q\prime} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q\prime} \tag{8}$$

where τ is the soft update rate.

3.4. Session classifier

In the session classifier, we adopt a Deep Neural Network(DNN) architecture to predict the class of session. The DNN network has an input layer, some fully connected layers and a non-liner layer from which the representation is used for session classification.

Input Layer

When a session is input to the feature selector, we will get two vectors: one is the current feature distribution vector of the session formulated as $s_i = \{f_1, f_2, \ldots, f_n\}$, the other is its feature mask formulated as $a_i = \{m_1, m_2, \ldots, m_n\}$. We multiply the corresponding positions of these two vectors to obtain a new feature distribution vector $s\prime_i = \{f\prime_1, f\prime_2, \ldots, f\prime_n\}$, where the value of retained features will remain unchanged, and the value of discarded features will become 0. We use this new feature distribution $s\prime_i$ as the input to DNN for classification.

Deep Neural Network

We found that few scholars have used DNNs directly for crawler detection through our research. Here we use five fully connected layers to map the feature distribution to a 2-dimensional vector and use softmax activation to represent the probability that a session is a crawler or a user. The probability for session classification $P(c|s\prime_i, \theta^C)$ is given as follows:

$$P(c|s\prime_i, \theta^C) = softmax(W_C * s\prime_i + b_C) \tag{9}$$

where $\theta^C = \{W_C, b_C\}$ is the parameter of session classifier.

Loss Function

Given the mini-batch training set S with N sessions, we define the objective function of the session classifier using cross-entropy as follows:

$$L_C = -\frac{1}{N}\sum_i logP(c_g|s\prime_i, \theta^C) \tag{10}$$

Model Train

In our approach, the result of the session classifier will affect the policy of the feature selector as a reward, and the result of the selector will affect the classification accuracy of the classifier, so the two modules should be trained jointly. The training process is described in Algorithm 1. To make the model converge as soon as possible and avoid unnecessary exploration, we need to pre-train the session classifier first, which is viewed as the environment of reinforcement learning, so that it can give relatively reliable rewards. We use a random process to generate some feature masks to retain and discard features. To make the training data more reliable and ensure a wide range of selection, we generate ten feature masks for each session. Each feature mask discards features according to different ratios, satisfying 0.0, 0.1, . . ., and 0.9 incremental rules. For example, ratio 0.0 means retaining all the features, and 0.9 means discarding 90 percent of the features. Experiments show that using such a pre-training approach, the session classifier can return a relatively reliable reward for each level of feature selection. Then, we fix the parameters of the session classifier and train the feature selector. At last, when the feature selector performs stably in the current environment, we jointly train the two modules.

Algorithm 1: Overall Training Process

1: Initialize the parameters of feature selector and session classifier with random weights respectively.
2: Pre-train the session classifier with random feature mask by minimizing Eq. (10)
3: Fix the parameters of session classifier and pre-train the feature selector by running Algorithm 2.
4: Train the two modules jointly until convergence.

The training process of the reinforcement learning model is shown in Algorithm 2. For each session in the training set, if the number of feature selection steps does not reach the upper limit N_t or the current feature distribution is insufficient to distinguish the session, which means it does not enter the terminal state, we will keep modifying its feature mask. When there is enough data in the replay buffer, we will sample a random mini-batch data to train the feature selector each time, and we will also train the session classifier if it is joint training. At the same time, we update the target network using soft update.

Algorithm 2: Reinforcement Learning Training Process

Input: Episode number N_e. Training data $S = \{s_1, s_2, \ldots, s_n\}$. A session classifier model C with parameters θ^C. A policy network μ with parameters θ^{μ} and a critic network Q with parameters θ^Q.

Initialize the parameters θ^{μ} and θ^Q.
Initialize the replay buffer M
Copy to get two target networks $\mu\prime$ and $Q\prime$ and initialize their parameters as: $\theta^{\mu\prime} = \theta^{\mu}, \theta^{Q\prime} = \theta^Q$.
for *episode* $e = 1$ *to* N_e **do**
  for $s_i \in S$ **do**
    for *step* $t = 1$ *to* N_t **do**
      Get an action a_i with policy network μ according to Eq. (1)
      State Transition: $s\prime_i \leftarrow s_i \times a_i$
      Get reward from session classifier: $r_i \leftarrow C(s\prime_i)$
      Add transition $(s_i, a_i, r_i, s\prime_i)$ into replay buffer M
      if $s\prime_i$ is terminal state **then**
        break
      end if
      $s_i = s\prime_i$ 15: **end for**
    Sample a random mini-batch B from M
    for all $(s_i, a_i, r_i, s\prime_i) \in B$ **do**
      Compute the loss of policy network and critic network respectively according to Eq. (5) and Eq. (6)
      Update the parameters θ^{μ} and θ^Q
      if *training jointly* **then**
        Compute the loss of session classifier according to Eq. (10).
        Update the parameters θ^C
      end if
      Soft update the parameters $\theta^{\mu\prime}$ and $\theta^{Q\prime}$ according to Eq. (7) and Eq. (8)
    end for
  end for
end for

4. Experiments

In this section, some experiments have been conducted over two real datasets to validate the proposed approach. The first is our own real dataset, which will be described in detail later, and the second is a public dataset from the search engine of the library and information center of the Aristotle University of Thessaloniki in Greece. We first demonstrated the existence of the diversity of crawlers in these two datasets experimentally. Then, to prove that our method can deal with the dynamic change of crawler, we compared our method with some baselines regarding the accuracy of the new type of crawler detection. We also did experiments on the accuracy of all types of crawlers detection. Finally, we looked at the effect of the number of the selected feature on the detection accuracy, and the experimental results and analysis will be given in the following text.

4.1. Datasets

4.1.1. Private dataset

Our dataset comes from the faculty personal home page portal of Zhejiang University. The site allows users to search faculty members' homepages, which contain personal information, photos, research directions, academic achievements, etc., based on their names or departments. We use the access log data of this website, and the time spans from June 22, 2020, to June 28, 2020, with a total of 2809702 requests.

- **Session Identification**
  Our session identification process works as follows. First, we group requests with the same IP address and user-agent string. In addition, we remove some requests with empty IP or empty user-agent. Then, we apply a timeout threshold to break the groups into sessions. In this paper, we set the timeout threshold to 30 min as in most studies. This process identified 56544 sessions which have an average of 49.69 requests.
  With a lot of research in the field of crawler detection, we selected 24 previously published features based on the characteristics of our dataset, the names and descriptions of the features are shown in Table 1.
- **Session Labeling**
  The session labeling process is a very complex task. To achieve a higher quality of labeling, we use multiple dimensions to label the session. First, we use a dataset from GitHub, which summarizes the common crawler user-agent [39]. It is a JSON file, and if the user-agent of our data is included in the JSON file, it will be labeled as a crawler; otherwise, it is temporarily labeled as user data.
  Second, we checked if the session contains a request to the *robots.txt* and a session will be labeled as a crawler if it contains such a request. This method is a standard labeling process because there are no external or internal links of *robots.txt* and only the crawler can access it.
  Third, We define some regular expressions to detect the user-agent, if the user-agent contains keywords such as a bot, spiders, spam, etc., it will be marked as a crawler.
  Finally, we define some features to label the crawlers regarding previous studies. For sessions where all requests do not access images, do not have a referrer, fail, or all requests are of HEAD type, we label them as crawlers. For the sessions that are not labeled as crawlers in any of the above dimensions, we label them as a user. We ended up with a private dataset containing 39581 user data and 16963 crawler data.

Table 1
Universal extracted features of private dataset.

ID	Feature name	Description
1	IS_TRAP_FILE	Whether to access trap file such as robots. txt.
2	NIGHT_RATIO	Percentage of requests made between 12am and 7am.
3	IMAGE_RATIO	Percentage of image file requests.
4	HTML_RATIO	Percentage of html file requests.
5	REFERRER_RATIO	Percentage of requests with unassigned referrer.
6	HEAD_RATIO	Percentage of requests of type HEAD.
7	304_RATIO	Percentage of requests with status code 304.
8	ERROR_RATIO	Percentage of erroneous requests.
9	ERROR_UPSTREAM_RATIO	Percentage of requests with empty upstream status.
10	SESSION_TIME	Total time elapsed between the first and the last request.
11	AVERAGE_INTERVAL	Average time between two consecutive requests.
12	DEVIATION_INTERVAL	Standard deviation of the time between two consecutive requests.
13	REQUESTS_NUMBER	The total number of requests.
14	UNIQUE_TYPE	The total number of file type of requests.
15	MAX_BROWSER_FILE_RATE	Maximum number of embedded resources in a web page.
16	PENALTY	Penalty for each backward and forward navigation or loop.
17	SD_RPD	Standard deviation of the page depth across all requests.
18	CSR	Percentage of requests with continuous access to the page belong to the same directory.
19	RES	Average response time to requests.
20	SF-FILE_TYPE	Switching factor of file types for each session.
21	SF-REFERRER	Switching factor on unassigned referrer.
22	WIDTH	The number of leaf nodes generated in the graph of all requests.
23	DEPTH	The maximum depth of the tree within the graph of all requests.
24	TOTA_PAGE	The total number of pages requested.

4.1.2. Public dataset

This dataset contains the server logs of the search engine of the Library and Information Center of the Aristotle University of Thessaloniki, Greece (http://search.lib.auth.gr/). Users can use this search engine to check the availability of books and other written works and search for digitized materials and scientific publications. This dataset contains access logs for an entire month from March 1 to March 31, 2018, including 4,091,155 requests. Through the publisher's session identification process and the labeling process, the access logs were divided into 67,352 sessions containing 53,858 user data and 13,494 crawler data.

4.2. Multi-type crawler analysis

This paper is based on the assumption that there are multi-types crawlers in the web environment and different types of crawlers have different characteristics and distributions in the feature space. To prove that our method can select the appropriate feature set for different types of crawlers, we should first verify whether the dataset satisfies the assumption of multi-types crawlers. Since both private and public datasets have only two types of labels, user or crawler, and there is no multi-type label for crawlers, we first use the K-means method to cluster the datasets and then use shap-value to analyze the differences in feature distributions between the clustering results. In addition, to present the

clustering results intuitively, we use the TSNE method to downscale the clustering results so that they have only two-dimensional features.

Parameter Setting: Through extensive experiments, for the parameters of the private dataset, we set the cluster number as 7, random state as 12; for the parameters of the public dataset, we set the cluster number as 6, random state as 12.

Result: The clustering results of the private dataset are shown in Fig. 3, where the green part is the user data, and the other colors are the crawler data. Although using the TSNE method to reduce the 24-dimensional feature space to a 2-dimensional space will cause an inevitable error, the individuals with similar feature spaces are still close together in the figure, and here we set the number of clusters to 7 to make each class basically contain only one kind of label and minimize the number of clusters.

We take out the six clustering results belonging to the crawler and perform a binary classification task with the user data respectively. To analyze the importance of different features in the classification process, here we used the xgboost classifier and analyzed the shap-value of different features using the shap toolkit. The analysis results are shown in Fig. 4. From the results, we can see that different clustering results have different feature importance when classifying with user data. Take the second and fourth categories for comparison, where the image ratio of the second category crawler has a significant difference with the user data, so the IMAGE_RATIO feature has a high shap-value, and the use of this feature can better distinguish crawler from user data. The fourth type of crawler has a significant difference between the TOTAL_PAGE with the user data, but on the contrary, there is basically no difference in the IMAGE_RATIO, so the shap-value of the TOTAL_PAGE is high, while the shap-value of the IMAGE_RATIO is very low. For this type of crawler, using the feature of TOTAL_PAGE can better distinguish crawler from user data.

The clustering results of the public dataset are shown in Fig. 5, where the blue part is the user data, and the other colors are the crawler data. From the clustering results, we can still see that different types of crawlers have apparent differences in feature distribution. Similarly, we analyzed the different crawler clustering results with users for feature importance, and the results are shown in Fig. 6. The results show that the features that have the highest shap-value vary for each type of crawler. The first type of crawler differs from the user data in the TOTAL_HTML feature, and it should be a crawler that only visits html files. In contrast, the second type of crawler differs from the user data in the IMAGES feature, and it should be an image crawler that only crawls images as mentioned before. The fourth type of crawler differs from the user data in the NIGHT feature, and it should be a crawler that only visits the website at night.

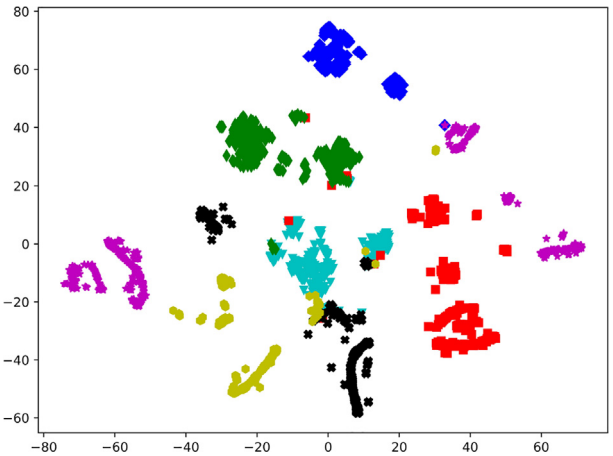


Fig. 3. Clustering results of the private dataset. The green part is the user data, and the other colors are the crawler data.

From the above experiments and analysis, we can conclude the following: firstly, the datasets we used have different types of crawlers. Secondly, different crawlers have different characteristics and distributions in the feature space.

Therefore, we can improve the accuracy of crawler detection by selecting different feature sets for multi-type crawlers.

4.3. Evaluation metrics

To show the effectiveness of the proposed method, we use the following metrics for evaluation. In addition to the commonly used recall, precision, F1-score and accuracy for classification tasks, we also add an evaluation metric for the white sample mishap, because the crawler detection problem has to ensure a low error rate in the actual business, which is responsible for seriously affecting the user's online experience.

- Recall: Given by $\frac{TP}{TP+FN}$, which represents the ratio of true positive to true positive plus false negative.
- Precision: Given by $\frac{TP}{TP+FP}$, which represents the ratio of true positive to true positive plus false positive.
- Error: Given by $\frac{FP}{FP+TN}$, which represents the ratio of false positive to false positive plus true negative.
- Accuracy: Given by $\frac{TP+TN}{TP+FP+TN+FN}$, which represents the ratio of true prediction to all the data.
- F1-score: Given by $\frac{2*P*R}{P+R}$, which considers both precision and recall in a single metric by taking their harmonic mean, where P and R are precision and recall respectively.

4.4. New type crawler detection

As mentioned before, crawlers have problems with diversity and dynamic change, the web environment is complex and volatile, and the emergence of new types of crawlers can make pre-trained models less robust. To check the robustness of our method in the face of new types of crawlers, we do the following experiments. Each time, we extract one type of crawler from the clustering results and mark it as a new type crawler. This part of data is not involved in the training process. After the model is trained, we test the robustness of the model by forming a test data set with user data and new type crawler, and observe the crawler detection effect of the model on this test set. For the private dataset, we have 6 types of crawlers, and we did experiments for each type separately, for a total of 6 sets of experiments. Similarly, for the public dataset we did 5 sets of experiments. We compare our method with some previous models that perform well.

Parameter Setting: For the parameters of the feature selector, we set the train episodes as 10, the step number of the agent as 10, learning rate of the policy network as 5e-6, learning rate of the critic network as 1e-5, gamma as 0.9, soft update rate as 0.01, feature selecting threshold as 0.4/0.6, terminal state threshold as 0.9, size of replay buffer as 5000, alpha as 10, batch size as 32, the delay coefficient τ as 0.01. For the parameters of the session classifier, we set the learning rate as 1e-3, and batch size as 32.

Baselines:

- DTMC: This method is based on a first-order discrete time Markov Chain model, which represents robot and human resource request patterns. They use this model to compute the probability a request pattern is more likely generated by a crawler or a user [10].

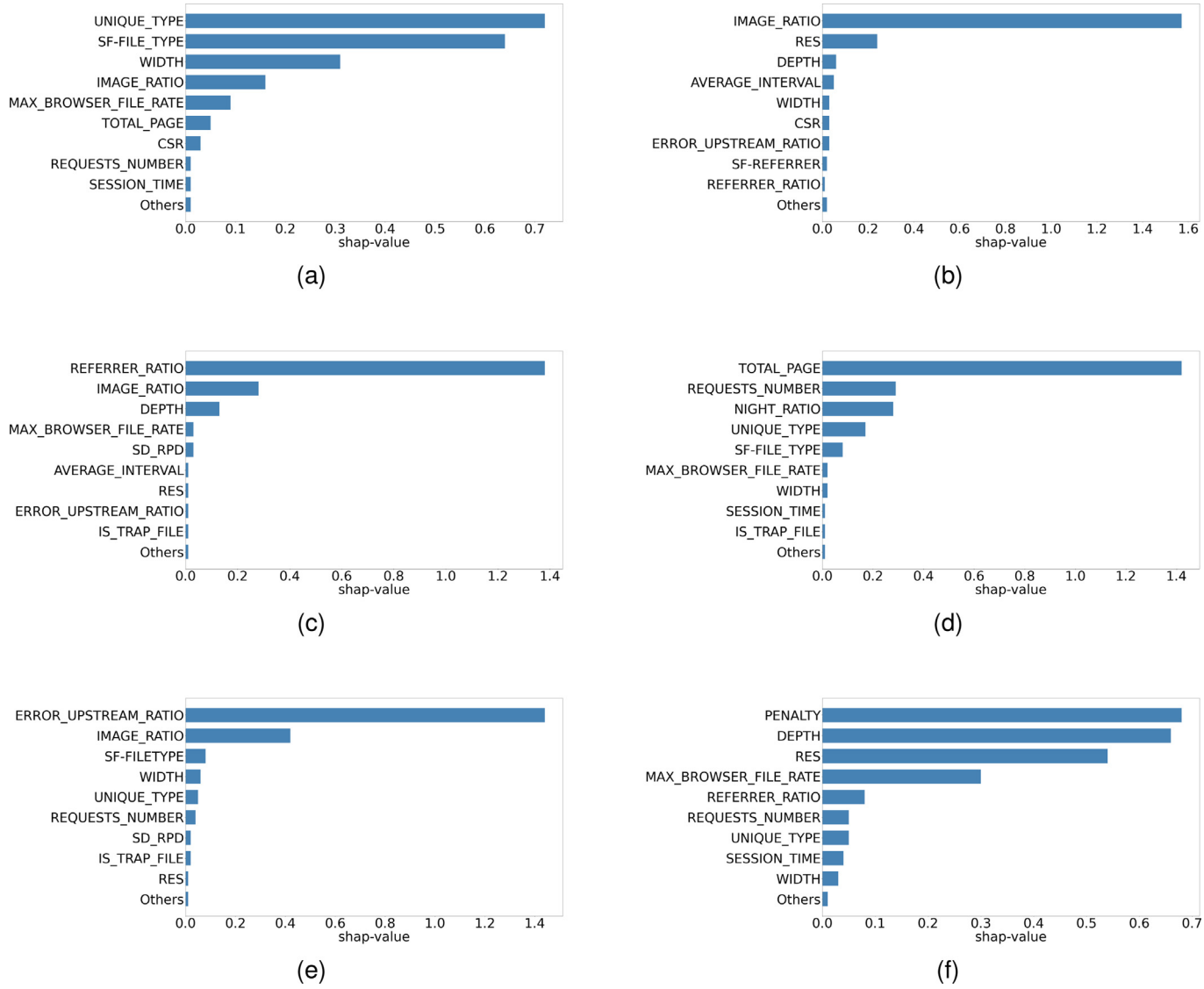


Fig. 4. The feature importance of each clustering result to user data in the private dataset. Each subgraph represents a comparison between a crawler clustering result with user data.

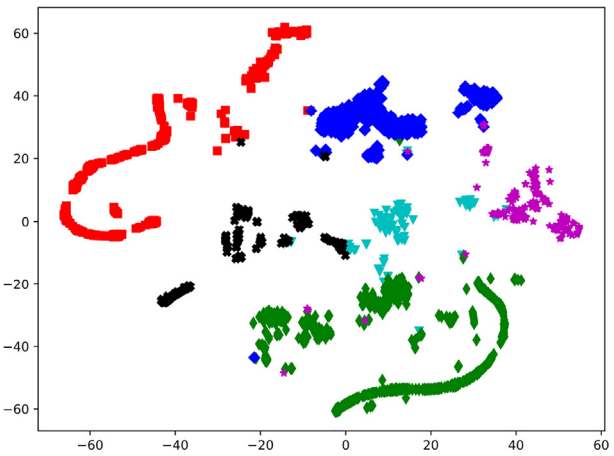


Fig. 5. Clustering results of the public dataset. The blue part is the user data, and the other colors are the crawler data.

- FRS_SOM: This model consists of two parts, the feature selecting module using the fuzzy rough set algorithm and the classification module using the SOM algorithm [13].
- MLP_SPRT: This approach uses deep neural networks combined with Wald's Sequential Probability Ratio Test to express the relationship between subsequent HTTP requests in an ongoing session and to assess the likelihood of each session being generated by a bot or human before it ends [40].
- BNC: This method constructs a Bayesian network that automatically classifies access log sessions as crawlers or users. They apply machine learning techniques to determine the parameters of the probabilistic model. The resulting classification is based on the maximum posterior probability of two classes [41].

Result: After a series of tuning parameters, we made each model show its best results. We used Recall, Precision, F1-score and Error rate as the judges of model effectiveness, and we first conducted experiments on the private dataset. The experimental results are shown in Fig. 7. The experimental results show that our model has the best performance in all metrics. DTMC has the worst per-

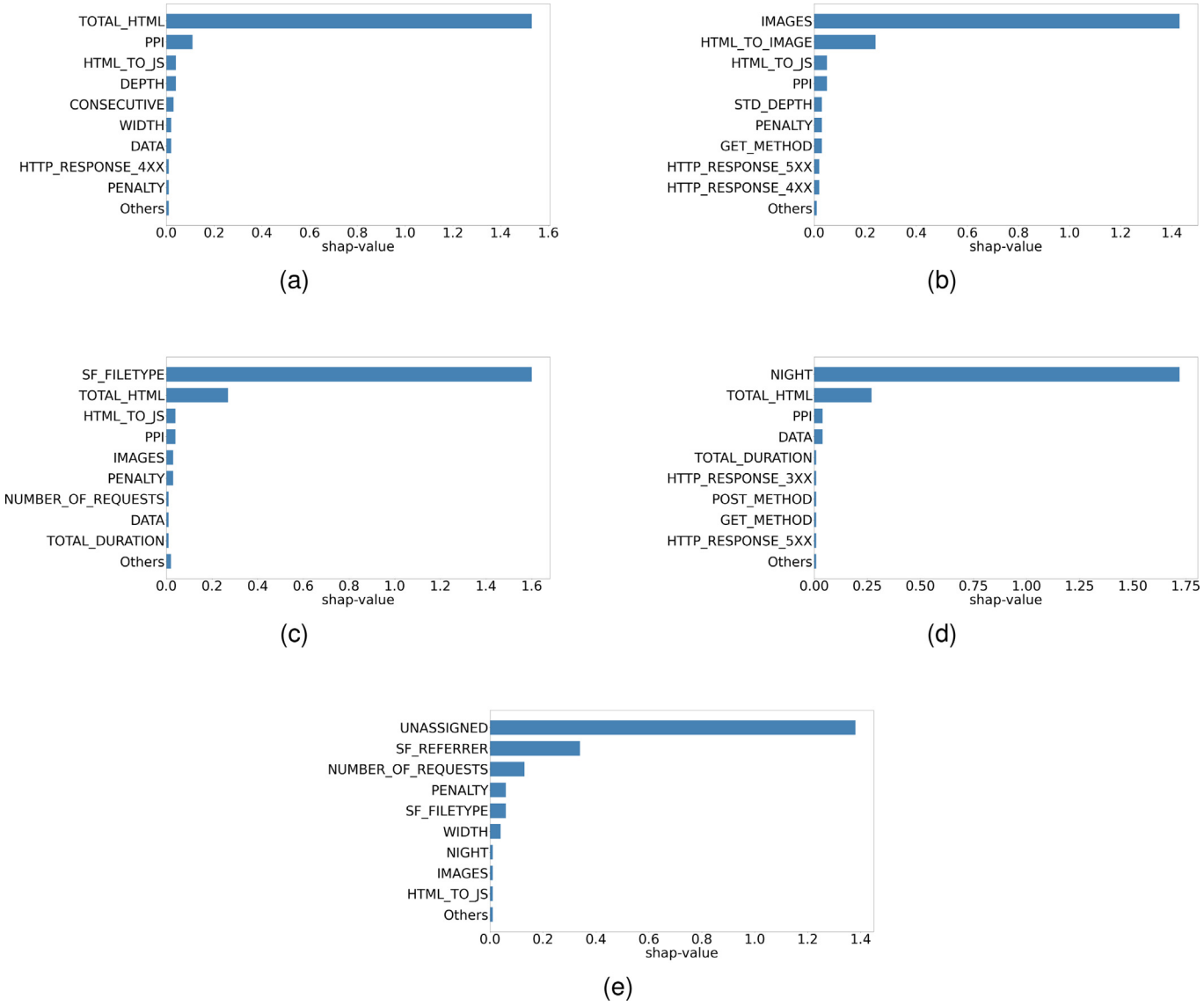


Fig. 6. The characteristic importance of each clustering result to user data in public dataset.

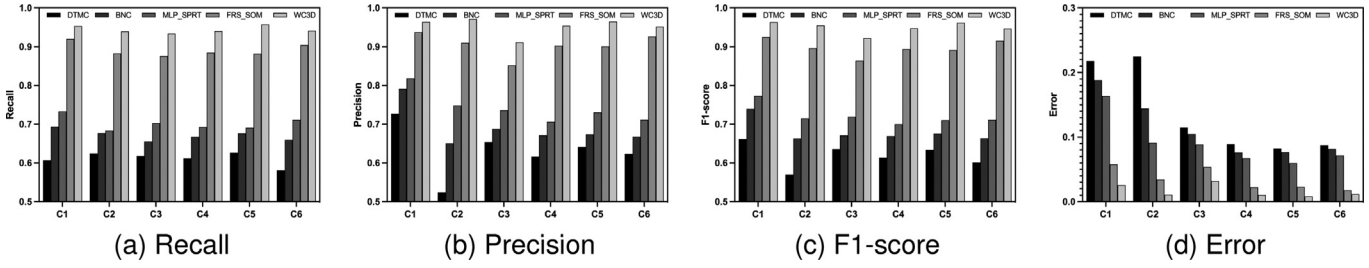


Fig. 7. The result of new type crawler detection for private dataset.

formance, considering that the private dataset does not have a complex file structure, and there are no frequent requests for different files on the website, so this method, based on the resource request patterns, does not perform well. BNC also performs poorly because it uses only six features, which can not represent the characteristics of the new type of crawler. MLP_SPRT performs relatively well because all the features are used, and the characteristics of the new crawler can be retained, but the recognition effect is not particularly good on account of a large number of redundant features and interference features. FRS_SOM performs well, which indicates that its feature filtering method filters out certain features that do not vary much across crawler types, allowing the model to detect some of the new types of crawlers. Our method performs better than FRS_SOM and can detect more new types of crawlers, thanks to the exploration process of reinforcement learning, where the feature selection module generates some

new feature combination patterns through exploration, enabling the model to adapt to the unknown environment to a certain extent, in other words, it still has some detection ability when facing the new type of crawlers. Similarly, we have done experiments on the public dataset, and the results are shown in Fig. 8. From the experimental results, it can be seen that our method achieves the best results. Considering that the websites used in the public dataset are more complex and the performance of each model is somewhat degraded compared to the private dataset, we do not experiment with DTMC because we do not have access to the characteristics of the resource request patterns of the public dataset.

4.5. Crawler detection accuracy

We have proved that our model has a more robust performance on the new type of crawler detection problem through experiments. We also want to verify the inherent detection accuracy of the model without considering the crawler dynamic change problem. We add an unsupervised baseline: DBC_WRD. The four features they used were Trap file request, Maximum rate of browser file request, Penalty and Percentage of 304 response codes, where Penalty and Maximum rate of browser file request are newly proposed. The experimental results show that the method in our paper has a more outstanding performance on both datasets.

Result: The results for the private and public datasets are shown in Table 2 and Table 3.

The experimental results show that our model has the best performance, with an accuracy rate of 0.99. DTMC still has the worst performance. In addition to our method, the FRS_SOM method and the MLP_SPRT method also perform well. The former uses a feature selecting method to filter out low-importance features to achieve better classification results, proving the effectiveness of feature selection. The latter uses all features and a neural network as the classifier. Only the FRS_SOM method uses the feature selecting method among these five baselines, but compared with the uniqueness of the selecting results of the fuzzy rough set method, our method will select different results that are suitable for different types of crawlers, which is one of the reasons why our method can get a higher accuracy rate. The DBC_WRD method does not perform well on the public dataset, considering that the method uses only four features for classification, and these four features on this dataset are less critical, so the classification result is not good, which again illustrates the need for feature selection.

4.6. Feature selection baseline

To demonstrate the effectiveness of our proposed feature selection method using reinforcement learning, we compared it with other feature selection methods. We conducted experiments using DNN as downstream classifiers in combination with different feature selection methods. For those methods that require a specified number of features, we uniformly set the number to the average of the number of features selected by our method.

Baselines:

- K-Best-Selection: This method uses label information and $\chi^2$to rank features and selects the top k features to form a feature set [42].
- mRMR: This method first ranks the features by minimizing their redundancy, while maximizing their relevance to the labels, and then selects the top k features [43].
- Recursive Feature Elimination (RFE): This method discards the features step by step in a recursive manner. First, the predictor is trained through all the features and the predictor scores the importance of each feature. After that, the least important features are deselected. This process is cycled until the required number of features are selected [44].
- FRS: this method selects features by FRS algorithm, which can consider the similarity of all features simultaneously and deal with the ambiguity of the data effectively [13].

Result: The results for the private and public datasets are shown in Table 4 and Table 5. The experimental results show that the feature set selected by our feature selection method is more effective in characterizing different types of crawlers, while other methods perform poorly because the selected feature set is single and separated from the downstream classifier

4.7. Screening ratio analysis

As mentioned before, we set threshold t_1 to map the weight w_i of each feature output from the policy network to feature mask m_i, where $w_i \in (0, 1)$ and $m_i \in \{0, 1\}$. If w_i is greater than t_1, the corresponding feature mask m_i is 1, otherwise it is 0. The setting of this parameter affects the number of retained features, which affects the accuracy of crawler detection. Therefore, we conducted a detailed experiment on this parameter to analyze the relationship between the number of retained features and the accuracy of crawler detection.

The results of the private dataset are shown in Fig. 9(a), where the red line shows the variation of classification accuracy with t_1, and the green line shows the variation of the average number of retained features of all sessions with t_1. From the figure, we can see that if we set t_1 to 0, the model will retain all features, at this time, the feature selector will no longer work, the model only has the session classifier module, which is equivalent to direct use of DNN for crawler detection, and the accuracy can reach 0.91. With the increase of t_1, more and more features are screened out, and the classification accuracy gradually increases. In this process, some features that will interfere with crawler detection are screened out, while those features that are more distinct between users and crawlers are retained. When the number of screened features reaches 6, the crawler detection accuracy reaches the highest 0.99. After that, as t_1 continues to increase, more features are screened out, and some features that help classify crawlers and users are also screened out, and the accuracy of the model starts

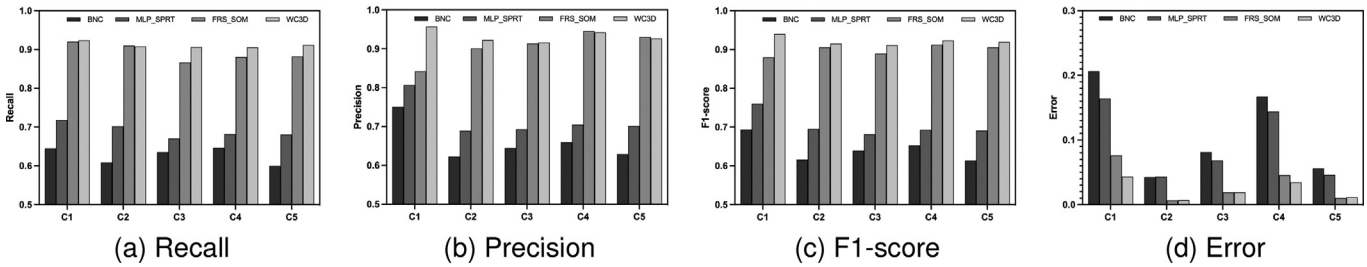


Fig. 8. The result of new type crawler detection for public dataset.

Table 2
The result of crawler detection for private dataset.

Method	Recall	Precision	F1	Acc	Error
DBC_WRD	0.9269	0.9660	0.9461	0.9683	0.0192
DTMC	0.6835	0.8634	0.7630	0.9244	0.0475
FRS_SOM	0.9647	0.9926	0.9784	0.9870	0.0109
MLP_SPRT	0.9621	0.9867	0.9742	0.9847	0.0113
BNC	0.9014	0.9942	0.9455	0.9684	0.0192
WC3D	**0.9982**	**0.9991**	**0.9987**	**0.9992**	**0.0034**

Table 3
The result of crawler detection for private dataset.

Method	Recall	Precision	F1	Acc	Error
DBC_WRD	0.9105	0.4034	0.5591	0.7301	0.2543
DTMC	/	/	/	/	/
FRS_SOM	0.8971	0.8519	0.8730	0.9490	0.0367
MLP_SPRT	0.8984	0.8785	0.8883	0.9550	0.0295
BNC	0.8851	0.6146	0.7254	0.8632	0.0845
WC3D	**0.9282**	**0.8788**	**0.9028**	**0.9607**	**0.0234**

Table 4
The result of feature selection analysis for private dataset.

Method	Recall	Precision	F1	Acc	Error
KBS	0.8117	0.8023	0.8069	0.8835	0.0857
mRMR	0.8439	0.8472	0.8456	0.9075	0.0652
RFE	0.8991	0.9107	0.9048	0.9433	0.0378
FRS	0.9332	0.9436	0.9384	0.9632	0.0239
WC3D	**0.9982**	**0.9991**	**0.9987**	**0.9992**	**0.0034**

Table 5
The result of feature selection analysis for public dataset.

Method	Recall	Precision	F1	Acc	Error
KBS	0.7743	0.7693	0.7718	0.9083	0.0582
mRMR	0.8142	0.8371	0.8255	0.9310	0.0397
RFE	0.8688	0.8840	0.8764	0.9509	0.0286
FRS	0.9178	0.8762	0.9019	0.9545	0.0238
WC3D	**0.9282**	**0.8788**	**0.9028**	**0.9607**	**0.0234**

to decrease. Until the number of retained features reaches 11, the model's accuracy drops to 0.87. The results of the public dataset are shown in Fig. 9(b). As with the private dataset, in the public dataset, the number of retained features decreases as t_1 increases, while the accuracy rate first increases and then decreases. When t_1 is 0, the model retains all features in the dataset, and the crawler

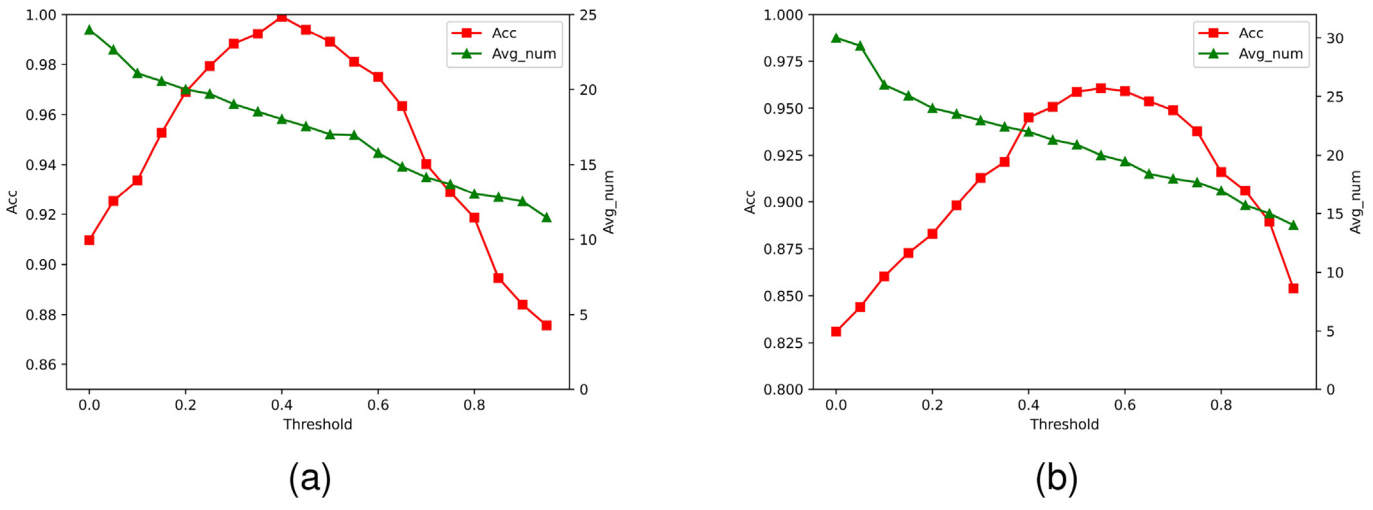


Fig. 9. The relationship between model accuracy and number of retained features. The result of the private dataset is shown in (a) and the result of the public dataset is shown in (b).

detection accuracy is 0.83. When t_1 is 0.55, the model filters out 10 features, and the accuracy reaches the highest 0.96. And when t_1 reaches the maximum, the model screened out 16 features, and the accuracy is only 0.85.

Based on the above experimental results, we can conclude that there are features in the dataset that can negatively affect the accuracy of crawler detection, and screening out these features will help distinguish crawlers from users. Second, by setting a threshold t_1, we can make the model screen out insufficient features to characterize a specific type of crawler and keep the good features to maximize the model's accuracy.

5. Conclusion

This paper proposes a new crawler detection method based on reinforcement learning for diversity and dynamics. The model consists of two modules: feature selector and session classifier. The feature selector module uses the DDPG architecture, consisting of a policy network and a critic network. This module is responsible for maximizing the reward by selecting suitable feature sets for different types of crawlers. It uses the feature distribution of the session as the state, the feature selection result as the action, and the session classification result as the reward. The session classification module uses DNN architecture. This module takes the retained features as input, outputs its classification results, and provides reward feedback to the feature selector based on the accuracy of the classification results. The two modules are trained jointly to achieve the best classification results.

Through experiments, we first demonstrate the existence of crawlers' diversity in the two datasets we used. Then we compare our method with state-of-the-art methods for the model's robustness when the new type of crawler appears, and our method achieves the best experimental results on both datasets. The results demonstrate the importance of feature selection and also that different types of crawlers have different feature distributions, and our method can select the appropriate set of representational features for crawlers based on their types. To sum up, our method has better performance in the complex network environment with crawlers' diversity and dynamics.

CRediT authorship contribution statement

Yang Gao: Conceptualization, Methodology, Software, Writing – original draft. **Xiaoyang Wang:** Data curation, Investigation. **Zunlei Feng:** Writing - review & editing. **Mingli Song:** Supervision. **Xingen Wang:** Project administration. **Xinyu Wang:** Formal analysis, Visualization. **Chun Chen:** Funding acquisition.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

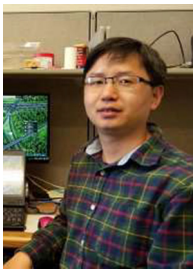
[1] P.-N. Tan, V. Kumar, Discovery of web robot sessions based on their navigational patterns, in: Intelligent Technologies for Information Analysis, Springer, 2004, pp. 193–222.
[2] H.N. Rude, D. Doran, Request type prediction for web robot and internet of things traffic, in: 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), IEEE, 2015, pp. 995–1000.
[3] I. Zeifman, Bot traffic report 2016. in Imperva Incapsula, 2017.
[4] C.L. Giles, Y. Sun, and I.G. Councill, Measuring the web crawler ethics, in Proceedings of the 19th international conference on World wide web, 2010, pp. 1101–1102.
[5] V. Almeida, D. Menascé, R. Riedi, F. Peligrinelli, R. Fonseca, and W. Meira Jr, Analyzing web robots and their impact on caching, in Proc. Sixth Workshop on Web Caching and Content Distribution, 2001, pp. 20–22.
[6] M.D. Dikaiakos, A. Stassopoulou, L. Papageorgiou, An investigation of web crawler behavior: characterization and metrics, Comput. Commun. 28 (8) (2005) 880–897.
[7] S. Ye, G. Lu, and X. Li, Workload-aware web crawling and server workload detection, in Proceedings of the second Asia-Pacific advanced network research workshop. Citeseer, 2004, pp. 263–269.
[8] S. Brin, L. Page, The anatomy of a large-scale hypertextual web search engine, Computer networks and ISDN systems 30 (1–7) (1998) 107–117.
[9] A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke, S. Raghavan, Searching the web, ACM Transactions on Internet Technology (TOIT) 1 (1) (2001) 2–43.
[10] D. Doran, S.S. Gokhale, An integrated method for real time and offline web robot detection, Expert Syst. 33 (6) (2016) 592–606.
[11] G. Suchacka, I. Motyka, Efficiency analysis of resource request patterns in classification of web robots and humans, ECMS (2018) 475–481.
[12] M. Zabihi, M.V. Jahan, J. Hamidzadeh, A density based clustering approach for web robot detection, in: 2014 4th International Conference on Computer and Knowledge Engineering (ICCKE), IEEE, 2014, pp. 23–28.
[13] J. Hamidzadeh, M. Zabihimayvan, R. Sadeghi, Detection of web site visitors based on fuzzy rough sets, Soft. Comput. 22 (7) (2018) 2175–2188.
[14] D. Doran, S.S. Gokhale, Web robot detection techniques: overview and limitations, Data Min. Knowl. Disc. 22 (1) (2011) 183–210.
[15] T. Kabe, M. Miyazaki, Determining www user agents from server access log, in: Proceedings Seventh International Conference on Parallel and Distributed Systems: Workshops, IEEE, 2000, pp. 173–178.
[16] P. Huntington, D. Nicholas, H.R. Jamali, Web robot detection in the scholarly information environment, J. Inf. Sci. 34 (5) (2008) 726–741.
[17] S. Kwon, Y.-G. Kim, S. Cha, Web robot detection based on pattern-matching technique, J. Inf. Sci. 38 (2) (2012) 118–126.
[18] S. Kwon, M. Oh, D. Kim, J. Lee, Y.-G. Kim, S. Cha, Web robot detection based on monotonous behavior, Proc. Inf. Sci. Ind. Appl. 4 (2012) 43–48.
[19] Q. Bai, G. Xiong, Y. Zhao, L. He, Analysis and detection of bogus behavior in web crawler measurement, Proc. Comput. Sci. 31 (2014) 1084–1091.
[20] F. Quan-Long, Y. Bin, Y. Zhou-Hua, X. Lei, Spider detection based on trap techniques, J. Comput. Appl. 30 (07) (2010) 1782.
[21] D. Doran, K. Morillo, and S.S. Gokhale, A comparison of web robot and human requests, in Proceedings of the 2013 IEEE/ACM international conference on advances in social networks analysis and mining, 2013, pp. 1374–1380.
[22] M. Motoyama, B. Meeder, K. Levchenko, G.M. Voelker, and S. Savage, Measuring online service availability using twitter, in 3rd Workshop on Online Social Networks (WOSN 2010), 2010.
[23] G. Jacob, E. Kirda, C. Kruegel, and G. Vigna, {PUBCRAWL}: Protecting users and businesses from {CRAWLers}, in 21st USENIX Security Symposium (USENIX Security 12), 2012, pp. 507–522.
[24] A. Lagopoulos, G. Tsoumakas, G. Papadopoulos, Web robot detection: A semantic approach, in: 2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI), IEEE, 2018, pp. 968–974.
[25] Y. Hiltunen, M. Lappalainen, Automated personalisation of internet users using self-organising maps, in: International Conference on Intelligent Data Engineering and Automated Learning, Springer, 2002, pp. 31–34.
[26] W. Zhu, H. Gao, Z. He, J. Qin, B. Han, A hybrid approach for recognizing web crawlers, in: International Conference on Wireless Algorithms, Systems, and Applications, Springer, 2019, pp. 507–519.
[27] X. Li, J. Ren, MICQ-IPSO: an effective two-stage hybrid feature selection algorithm for high-dimensional data, Neurocomputing 501 (2022) 328–342, https://doi.org/10.1016/j.neucom.2022.05.048 [Online]. Available:.
[28] A. Tan, J. Liang, W. Wu, J. Zhang, L. Sun, C. Chen, Fuzzy rough discrimination and label weighting for multi-label feature selection, Neurocomputing 465 (2021) 128–140, https://doi.org/10.1016/j.neucom.2021.09.007 [Online]. Available:.
[29] H.E. Kiziloz, Classifier ensemble methods in feature selection, Neurocomputing 419 (2021) 97–107, https://doi.org/10.1016/j.neucom.2020.07.113 [Online]. Available:.
[30] T. Gržinić, L. Mršić, J. Šaban, Lino-an intelligent system for detecting malicious web-robots, in: Asian Conference on Intelligent Information and Database Systems, Springer, 2015, pp. 559–568.
[31] M. Zabihi, M. Vafaei Jahan, and J. Hamidzadeh, A density based clustering approach to distinguish between web robot and human requests to a web server, The ISC International Journal of Information Security, vol. 6, no. 1, pp. 77–89.
[32] S. Fan, X. Zhang, Z. Song, Reinforced knowledge distillation: Multi-class imbalanced classifier based on policy gradient reinforcement learning, Neurocomputing 463 (2021) 422–436, https://doi.org/10.1016/j.neucom.2021.08.040 [Online]. Available:.
[33] Y. Li, Y. Fang, Z. Akhtar, Accelerating deep reinforcement learning model for game strategy, Neurocomputing 408 (2020) 157–168, https://doi.org/10.1016/j.neucom.2019.06.110 [Online]. Available:.
[34] J. Janisch, T. Pevný, and V. Lisý, Classification with costly features using deep reinforcement learning, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, no. 01, 2019, pp. 3959–3966.
[35] Z. Xu, Y. Wang, J. Jiang, J. Yao, L. Li, Adaptive feature selection with reinforcement learning for skeleton-based action recognition, IEEE Access 8 (2020) 213038–213051.

[36] J. Feng, M. Huang, L. Zhao, Y. Yang, and X. Zhu, Reinforcement learning for relation classification from noisy data, in: Proceedings of the aaai conference on artificial intelligence, vol. 32, no. 1, 2018.
[37] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, Continuous control with deep reinforcement learning, arXiv preprint arXiv:1509.02971, 2015.
[38] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, M. Riedmiller, Deterministic policy gradient algorithms, in International conference on machine learning, PMLR (2014) 387–395.
[39] OzzyCzech, crawler-user-agents, https://github.com/monperrus/crawler-user-agents, 2021.
[40] A. Cabri, G. Suchacka, S. Rovetta, F. Masulli, Online web bot detection using a sequential classification approach, in: 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), IEEE, 2018, pp. 1536–1540.
[41] A. Stassopoulou, M.D. Dikaiakos, Web robot detection: A probabilistic reasoning approach, Comput. Netw. 53 (3) (2009) 265–278.
[42] Y. Yang and J.O. Pedersen, A comparative study on feature selection in text categorization, 1997.
[43] H. Peng, F. Long, C. Ding, Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy, IEEE Trans. Pattern Anal. Mach. Intell. 27 (8) (2005) 1226–1238.
[44] P.M. Granitto, C. Furlanello, F. Biasioli, F. Gasperi, Recursive feature elimination with random forest for ptr-ms analysis of agroindustrial products, Chemometrics Intell. Lab. Syst. 83 (2) (2006) 83–90.

Yang Gao received the master degree in computer science from Zhejiang University of China, in 2017. He is currently pursuing the Ph.D. degree with the College of Computer Science, Zhejiang University. His research interests include anomaly detection and timeseries data mining.

Xiaoyang Wang is a master student in Computer Technology from College of Computer Science, Zhejiang University, and received his B.Eng. Degree in Computer Science and Technology from Zhejiang University. His research interests mainly include reinforcement learning, deep learning, machine learning.

Zunlei Feng is an assistant research fellow in College of Software Technology, Zhejiang University. He received his Ph.D degree in Computer Science and Technology from College of Computer Science, Zhejiang University, and B. Eng. Degree from Soochow University. His research interests mainly include computer vision, image information processing, representation learning, medical image analysis. He has authored and co-authored many scientific articles at top venues including IJCV, NeurIPS, AAAI, TVCG, ACM TOMM, and ECCV. He has served with international conferences including AAAI and PKDD, and international journals including IEEE Transactions on Circuits and Systems for Video TechnologyInformation Sciences, Neurocomputing, Journal of Visual Communication and Image Representation and Neural Processing Letters.



Mingli Song (M'06-SM'13) received the Ph.D. degree in computer science from Zhejiang University, China, in 2006. He is currently a Professor with the Microsoft Visual Perception Laboratory, Zhejiang University. His research interests include face modeling and facial expression analysis. He received the Microsoft Research Fellowship in 2004.

Xingen Wang received the graduate and PhD degrees in computer science from Zhejiang University of China, in 2005 and 2013, respectively. He is currently a research assistant in the College of Computer Science, Zhejiang University. His research interests include distributed computing and software performance.

Xinyu Wang received the graduate and PhD degrees in computer science from Zhejiang University of China, in 2002 and 2007, respectively. He was a research assistant at the Zhejiang University, from 2002 to 2007. He is currently a professor in the College of Computer Science, Zhejiang University. His research interests include streaming data analysis, formal methods, very large information systems, and software engineering.

Chun Chen is currently a Professor with the College of Computer Science, Zhejiang University. His research interests include computer vision, computer graphics, and embedded technology.